Papers and Presentations of the
15th Annual EICAR Conference
"Security in the Mobile and Networked World"

Edited by
Paul Turner and Vlasti Broucek
School of Information Systems, University of Tasmania, Australia

- Hamburg, Germany -
29 April - 2 May 2006

This document contains all research/academic papers that are included in the printed
Proceedings of the 15th Annual EICAR Conference
***"Security in the Mobile and Networked World"* (ISBN: 87-987271-8-4)**,
as well as majority of other papers and presentations accepted for the conference. However, only research/academic papers in the first section of this document were peer-reviewed.

Preface

EICAR2006 – 'Security in the Mobile and Networked World' is the 15th Annual EICAR Conference. This Conference (held from 29th April to 2nd May) at the Haffen Hotel, Hamburg, Germany brings together experts from industry, government, military, law enforcement, academia, research and end-users to examine and discuss new research, development and commercialisation in anti-virus, malware, computer and network security and e-forensics.

The continuing success of EICAR bears witness to the recognition amongst participants of the importance and benefit of encouraging interaction and collaboration between industry and academic experts from within the public and private sectors. As digital technologies become ever-more pervasive in society and reliance on digital information grows, the need for better integrated socio-technical solutions has become even more challenging and important.

This year EICAR2006 has again seen an increase in both the quality and quantity of papers. This made the conference committee's task of paper acceptance hard but enjoyable. To maximise interaction and collaboration amongst participants, two types of conference submissions were invited and subsequently selected – industry and research/academic papers. These papers were then organised according to topic area to ensure a strong mix of academic and industry papers in each session of the conference.

Industry papers were selected on the basis of the submission of abstracts while research/academic papers were selected after a rigorous blind review process organised by the program committee. Each submitted paper was reviewed by at least three members of the program committee with approximately one third of all submitted papers rejected. The quality of accepted papers was excellent and the organising committee is proud to announce that authors of several papers have already been invited to submit revised manuscripts for publication in a number of major journals.

From the papers submitted and accepted for this year's conference there is strong evidence to support the view that the EICAR conference is growing in its international reputation as a forum for the sharing of information, insights and knowledge both in its traditional domains of malware and computer viruses and also increasingly in critical infrastructure protection, intrusion detection and prevention and legal, privacy and social issues related to computer security and e-forensics.

Program Committee

We are grateful to the following distinguished researchers and/or practitioners (listed alphabetically) who had the difficult task of reviewing and selecting the papers for the conference:

Elizabeth Bates	Valid Technologies, USA
Dr Andrew Blyth	School of Computing, University of Glamorgan, UK
Vlasti Broucek (EICAR Scientific Director and Program Chair)	School of Information Systems, University of Tasmania, Australia
Dr Hervé Debar	France Télécom Research and Development, France
Professor Eric Filiol	Laboratoire de virologie et de cryptologie, Ecole Supérieure et d'Application des Transmissions, Rennes, France
Professor Richard Ford	Florida Institute of Technology, USA
Dr Steven Furnell	University of Plymouth, UK
Professor Urs E Gattiker	International School of New Media (ISNM), University of Luebeck, Germany
Assoc. Professor William (Bill) Hafner	Nova Southeastern University, USA
Assist. Professor Marko Helenius	University of Tampere, Finland
Dr Andy Jones	BT, UK
Professor Yves Poullet	Centre de Recherches Informatique et Droit (CRID), Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium
Professor Gerald Quirchmayr	University of Vienna, Austria University of South Australia, Australia
Sebastian Rohr	Computer Associates, Germany
Dr Jill Slay	Enterprise Security Management Laboratory, University of South Australia, Australia
Peter Sommer	London School of Economics, UK
Assoc. Professor Paul Turner (Proceedings Editor)	School of Information Systems, University of Tasmania, Australia
Christine Whalley, CISSP	Pfizer Inc., USA
Professor James Wolfe (EICAR Technical Director)	University of Central Florida, USA

Paul Turner and Vlasti Broucek
Editors
School of Information Systems
University of Tasmania
Australia

Email: [Paul.Turner@postoffice.utas.edu.au], [Vlasti.Broucek@utas.edu.au]

Contents

Peer Reviewed Papers

This section contains papers selected for the conference based on full peer review process by the program committee

Other Papers and Contributions

This section contains papers selected by the program committee based on submitted abstracts

Introduction

This year EICAR celebrates in style its 15th Annual Conference (from 29th April to 2nd May) at the Haffen Hotel, Hamburg, Germany. It is very pleasing to be able to highlight how this year's Conference bears witness to the way that EICAR has continued to grow from strength to strength since its inception. More significantly, it is a credit to the efforts of the program committee (particularly over the last five years) that EICAR has developed such a strong International reputation as one of the few high quality conferences able to successfully bridge industry, government and academia.

As this year's conference program and delegate list illustrates EICAR is continuing to attract a diverse range of experts engaged in new research, development and commercialisation in anti-virus, malware, computer & network security, and e-forensics. This year's conference theme 'Security in the Mobile and Networked World' draws our attention to the issues arising from the reality of an 'anytime, anywhere web'. In this context, it can be argued that there is an even stronger need for open forums where vigorous and rigorous interaction can occur amongst representatives from industry, government, military, law enforcement, academia, research and end-users. These proceedings are an excellent example of this diversity and clearly reveal the challenges arising from the convergence and clash of different streams of research, development and commercialisation. With papers addressing technical, organisational and socio-legal issues arising in the era of the wireless web it is evident that the on-going challenges of how to effectively balance the requirements for network security, individual privacy and the need for legally admissible digital evidence remains.

Please enjoy the papers published in these proceedings and I look forward to meeting you in the near future. I would also like to take this opportunity to actively encourage you to communicate and forge collaborations with EICAR. I look forward to your on-going participation in the EICAR conference and thank you for your contribution to its success.

Paul Turner
School of Information Systems
University of Tasmania
Australia

Email: [Paul.Turner@utas.edu.au]

Peer reviewed papers

A parallel 'String Matching Engine' for use in high speed network intrusion detection systems.

Gerald Tripp

University of Kent

About Author

Gerald Tripp is a Lecturer in Computer Science at the University of Kent.
Contact Details: The Computing Laboratory, University of Kent, Canterbury, Kent, CT2 7NF, UK, phone +44 1227 827566, fax +44 1227 762811, e-mail G.E.W.Tripp@kent.ac.uk

Keywords

Security, intrusion detection, finite state machine, string matching, high speed networks.

A parallel 'String Matching Engine' for use in high speed network intrusion detection systems.

Abstract

This paper describes a finite state machine approach to string matching for an intrusion detection system. To obtain high performance, we typically need to be able to operate on input data that is several bytes wide. However, finite state machine designs become more complex when operating on large input data words, partly because of needing to match the starts and ends of a string that may occur part way through an input data word.

Here we use finite state machines that each operate on only a single byte wide data input. We then provide a separate finite state machine for each byte wide data path from a multi-byte wide input data word. By splitting the search strings into multiple interleaved substrings and by combining the outputs from the individual finite state machines in an appropriate way we can perform string matching in parallel across multiple finite state machines.

A hardware design for a parallel string matching engine has been generated, built for implementation in a Xilinx Field Programmable Gate Array and tested by simulation. The design is capable of operating at a search rate of 4.7 Gbps with a 32-bit input word size.

Introduction

Network intrusion detection consists of monitoring computer networks for various types of security attack. This can be network wide monitoring (network based) or it can be at each individual host computer in the system (host based). Basic network security is provided by network firewalls, which act as an intermediary between the Internet and a local network – these filter network traffic on the basis of header fields in the packets such as the source and destination IP address and TCP port numbers. This type of filtering is good at blocking a large proportion of unwanted incoming traffic. However, some network attacks may be targeted at machines such as web and mail servers that need to be visible through the firewall. In this case, it may be necessary to look inside each incoming data packet to determine whether it represents a potential threat. We may then wish to block that traffic (intrusion prevention) or be able to generate an alert that potentially malicious traffic is present (intrusion detection). The problem is that we may have no particular field to examine inside the packet, and may need to search the entire packet. This is the standard technique that we use for intrusion detection: we first look at the header fields of the packet to see if the packet is potentially of interest and if so we then search the content of the packet for one or more related intrusion detection 'signatures'. These signatures are short search strings which are chosen as representing a high probability of an attack occurring when present, whilst having a low probability of occurring otherwise.

A lot of current intrusion detection systems are software based, the most well known example probably being Snort (Roesch, 1999). Software solutions can however have problems when presented with a high network load. One solution can be to use host based intrusion detection and to require each computer to perform its own intrusion detection. This however can be targeted by denial of service attacks to put the intrusion detection software on individual machines under heavy load. Host based solutions are also only possible if we are able to add intrusion detection software to each host system, and this may not be the case with some embedded systems.

Summary of this paper

This paper looks at the string matching part of intrusion detection and describes how it is possible to build a 'string matching engine' for implementation in a Field Programmable Gate Array (FPGA) that uses fine grained parallelism to improve its search rate. The method used is to operate on a multi byte input data word and to partition the matching operation between a set of Finite State Machines (FSMs), each of which processes one of the byte streams from a multi-byte wide network input and looks for parts of the search string. The results from these multiple FSMs are then combined in a particular way so as to determine whether a string has been matched across all the FSMs.

The next section describes the background and outlines some of the related work in this field. The following section describes the operation of the parallel string matching system proposed in this paper. The software section gives the results of processing multiple search strings and the resource requirements for various string set sizes and implementation options. The next section gives details of a hardware design for a string matching engine and its performance and resource requirements. The final section gives conclusions and ideas for further work.

Discussion

A lot of existing intrusion detection systems are software based, the most well known example probably being Snort (Roesch, 1999). Many improvements have been made to Snort by optimising the order in which data is compared. Work by (Kruegel & Toth, 2003) uses rule clustering and is implemented as a modified snort rule engine. This uses decision trees to reduce the number of comparisons made against incoming network data and uses a multiple string matching algorithm based on the work by (Fisk & Varghese, 2001).

A paper by (Abbes, Bouhoula & Rusinowitch, 2004) describes a system using a decision tree in conjunction with protocol analysis. The protocol analysis uses a specification file for the protocol being monitored and performs 'Aho-Corasick' (Aho & Corasick, 1975) string matching on only the appropriate parts of the data stream. This technique reduces the overall workload and also reduces the number of false positives as compared with performing matching on the entire data packet or using simple offset and depth constraints.

Work by (Paul, 2004) looks at distributed firewalls and implements stateful packet classification spread across consecutive firewalls. This helps to spread the workload between separate machines.

It can be difficult to perform intrusion detection in software at high network traffic rates and hardware solutions may be required. Software solutions being essentially sequential also suffer from performance problems as we increase the number of rules; (Cho & Mangione-Smith, 2004) state that a software system with 500 rules may have difficulty in sustaining a throughput of 100 Mbps. Hardware solutions have different limitations; we can often increase the number of rules without affecting throughput because of the use of parallelism – the cost of increasing the number of rules may be an increase in hardware resource utilisation instead.

Overview of existing solutions

A number of hardware based string matching systems for intrusion detection have been described in the literature; an overview of some of the techniques is given below.

A product called ClassiPi from PMC-Sierra is described by (Iyer, Kompella & Shelat, 2001), this is a classification engine and implemented as an application specific integrated circuit (ASIC). This

device allows software-like algorithms to be used for various packet classification and packet inspection operations, including the use of regular expressions to search the contents of packets.

Work by (Attig & Lockwood, 2005) uses Bloom filters to perform string searching. Bloom filters provide an efficient method to perform searching for a large number of strings in parallel, but suffer from the disadvantage of producing false positive matches. Attig and Lockwood show that Bloom filters can be used as a very efficient front end to remove the bulk of the network traffic that is known to be benign before input into a conventional software intrusion detection system.

(Cho et.al., 2004) describe a system that uses multiple matching systems, each of which will search incoming network data for a set of distinct string 'prefixes'. For each possible string prefix, their system will lookup the remaining part of the string that must be compared sequentially against the incoming data to determine whether that string is actually present. Multiple strings with identical prefixes need to be distributed between different matching systems.

An interesting approach is taken by (Baker & Prasanna, 2004), who have a series of input comparators for each data byte of interest – the output of these comparators each feed into a pipeline of flip-flops. Strings can be identified by the use of an AND function that looks for all the required data bytes for a string in the appropriate positions within the pipeline. They show that this can be extended to operate with multi-byte input data by the use of multiple sets of pipelines and looking for strings across the set of pipelines at all byte alignments.

Finite state machine approaches

A number of systems have been designed that use Finite State Machines (FSM) to perform the searching – most of these use a Deterministic Finite Automata (DFA) to implement string matching. This type of FSM has sets of states, inputs and outputs; the FSM can be in one of its states and there is a mapping between each pair of current state and input to the next state and output. When used in string matching, we use the FSM state to define how much of a string we have matched so far.

The approaches taken by (Sugawara, Inaba & Hiraki, 2004) and by (Tripp, 2005) is to first compress multi-byte input data into a number of different patterns that are of interest and then to use DFAs to perform string matching several bytes at a time. (Moscola, Lockwood, Loui & Pachos, 2003) convert regular expressions into DFA that operate one byte at a time and show that this can be used to perform matching for standard spam-assassin rules without creating too many DFA states.

A different approach is taken by (Franklin, Carver & Hutchings, 2002), who implement Non-deterministic Finite Automata (NFA) in hardware to perform matching of strings from the Snort rule set, this approach first being proposed by (Sidhu & Prasanna, 2001). This was extended by (Clark & Schimmel, 2004) to operate with multi byte input data.

The text by (Hopcroft, Motwani & Ullman, 2001) gives a comprehensive coverage of Deterministic and Non-deterministic Finite Automata.

String matching algorithms

There are many string matching algorithms described in the literature, most of which were originally devised for software implementation. A hardware implementation has slightly different requirements than that for a software implementation and may well need to be less complex. For efficiency it is more common to build systems that work on a stream of data, rather than providing random access to the contents of a buffer; ideally we would like the string matching to operate at a deterministic rate to avoid the need for buffering.

The fastest method of matching strings is considered to be the Boyer-Moore algorithm (Boyer & Moore, 1977) and its successors. This performs string matching on a 'right to left' basis and skips forward on a mismatch. This gives an average performance that is usually sub-linear, but a worst case performance that may require us to look at some input bytes many times.

The 'Knuth Morris Pratt' (KMP) algorithm (Knuth, Morris & Pratt, 1977), performs matching on a left to right basis and on mismatch will use the longest partial match as a starting point for further matching. The algorithm can be adapted to operate at deterministic data rate and not re-examine input data on a mismatch.

The Aho-Corasick algorithm (Aho et.al., 1975) matches several strings at the same time. This works by constructing a trie containing the various strings and this is traversed as the data arrives. As with KMP, this can also be modified to operate at a deterministic rate only looking at each input data item once.

Both KMP and Aho-Corasick can be implemented by creating a FSM that operates at one input data item per clock cycle and are therefore ideal for hardware implementation. A common method of implementation for both these algorithms uses a maximum FSM size of an initial state and one state per search character (in one or all strings). When using Aho-Corasick, we would have fewer states when common prefixes of search strings enable us to share a FSM state. The state transition information in both cases will vary in complexity determined by whether on mismatch of a partly matched string there exists a suffix of the data matched that forms a smaller partial match of that string (or another).

Parallel string matching

From the work presented by (Sugawara et.al., 2004) and (Tripp, 2005), we can see that high performance can be obtained by creating a FSM that will match multiple bytes in the same clock cycle. However this has the overhead of compressing the input data so as to present a small input word to the FSM. A second issue is that the start and ends of strings have a high chance of appearing part way through an input data word, so we may need to match parts of the start and end of a string with 'wild card' characters.

It is far easier to match data from an 8-bit input bus, but this does not give such good throughput. The solution proposed here is to use multiple finite state machines in parallel to process the input data. Course grained parallel FSM solutions have already been implemented, such as the work described by (Moscola et.al., 2003), where input packets are allocated to a number of content scanners on a round robin basis. We propose a fine-grained from of parallelism, where multiple finite state machines process each packet in parallel.

Parallel finite state machines

The approach we take here is to provide a finite state machine for each byte stream from a multi-byte input data word. If we have a w-byte wide input word, then we can use w separate finite state machines, each of which are looking for all w instances of the 'substrings' made up from a w-way interleave from the search string. An example of such a system is shown in Figure 1.

A related, but different, approach is taken by (Tan & Sherwood, 2005) who use multiple FSMs running in parallel to match a sequence of bits, with each FSM matching a particular bit position from the input data.

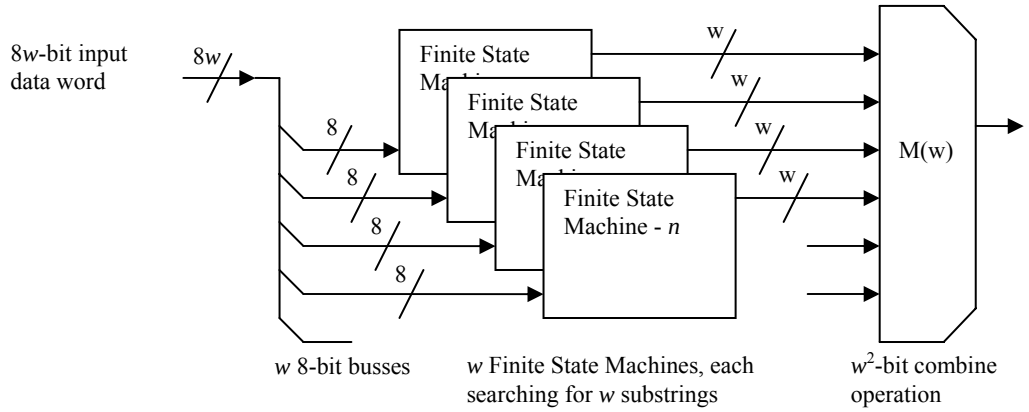


Figure 1 - Matching Interleaved substrings.

All w instances of our FSM are identical, and each will be looking for all w substrings. Each FSM has a w-bit Boolean 'match vector' output to specify the sub-strings matched in any clock cycle. If we find all w substrings appearing in an appropriate order across all w finite state machines at the correct time, then we will have found our search string. We can see an example of a set of substrings of a given search string when $w = 4$ in Figure 2.

Word size = 4

```
Search string =     "the-cat-sat-on-the-mat"
Substring 0 =       "  e   t   t   -   -   "      = "ett--"
Substring 1 =       "   -   -   -   t   m  "      = "---tm"
Substring 2 =       "t   c   s   o   h   a "      = "tcsoha"
Substring 3 =       " h   a   a   n   e   t"      = "haanet"
```

(The substrings are sorted by the order of completion, the reason for which will be explained below.)

Figure 2 - Interleaved substrings.

By sorting our substrings on the basis of the order of completion of the match, we have a sequence in byte terms of w consecutive substring matches. However, we are processing our data on the basis of a w-byte input word. The string may be aligned in one of w different ways, with the last w bytes occurring in one or two input data words – the occurrence of each of the last w bytes of the search string relate to the instant when each of the related substring matches will occur. We define here an alignment of c as meaning that of the last w bytes of the search string, c of these will occur in one input word, followed by $(w-c)$ in the following input word, where: $0 \le c < w$.

Byte stream x is being monitored by finite state machine x. Each of the finite state machines is searching for all w substrings, and has a Boolean 'match' output for each substring y. Thus we have a group of w^2 FSM outputs: O_{xy} where $0 \le x < w$ and $0 \le y < w$, relating to whether FSM x has detected substring y in the current clock cycle. We are also interested in whether string matches occurred in the previous clock cycle, and O'_{xy} is a delayed (pipelined) copy of O_{xy} from the previous clock cycle.

Taking the case where $w = 4$ and $c = 1$ for the string in Figure 2, we have the alignment shown in Table 1.

Input Byte	n-5	n-4	n-3	n-2	n-1	n
0		-	-	-	t	m $*_1$
1	t	c	s	o	h	a $*_2$
2	h	a	a	n	e	t $*_3$
3	e	t	t	-	- $*_0$	

Input word

$*_S$ indicates when a match occurs for substring S.

Table 1 - String match at alignment c=1.

We define $M_c(w)$ as being a Boolean operation specifying whether a match occurs at alignment c, in a system with a word size w. In our example above, we have $c = 1$ and $w = 4$; we can see from Table 1, that $M_1(4)$ is as shown in Equation 1.

$$M_1(4) = O'_{3\,0} . O_{0\,1} . O_{1\,2} . O_{2\,3}$$

Equation 1 - Match occurs at alignment c=1, for word size w=4.

This follows a very simple pattern, and we can produce a general formula for $M_c(w)$. Our complete string match is then defined as $M(w)$ which determines whether the match occurs at any of the w possible alignments. This is shown in Equation 2.

$$M_c(w) = \bigwedge_{i=0}^{w-1} \text{if}\,(i \geq c)\ \text{then}\ (O_{(i-c)\,i})\ \text{else}\ (O'_{(i+w-c)\,i})$$

$$M(w) = \bigvee_{c=0}^{w-1} M_c(w) = \bigvee_{c=0}^{w-1} \left(\bigwedge_{i=0}^{w-1} \text{if}\ (i \geq c)\ \text{then}\ \left(O_{(i-c)\ i}\right) \text{else}\ \left(O'_{(i+w-c)\ i}\right) \right)$$

Equation 2 – The Combine operation.

(Note that in Equation 2, we use $\bigvee$ and $\bigwedge$ to represent the Boolean 'inclusive-or summation' and 'and product' respectively.)

The combine operation $M(w)$ is independent of the search string and can be implemented as a fixed logic function for a given value of w. We also need $\sum_{x=1}^{w-1} x$ D-type flip flops to generate the delayed versions of some of the inputs. As an example, the combine operation required for a system with a word size of 4 bytes is shown in Equation 3.

$$M(4) = O_{0\,0} . O_{1\,1} . O_{2\,2} . O_{3\,3} + O'_{3\,0} . O_{0\,1} . O_{1\,2} . O_{2\,3} +$$
$$O'_{2\,0} . O'_{3\,1} . O_{0\,2} . O_{1\,3} + O'_{1\,0} . O'_{2\,1} . O'_{3\,2} . O_{0\,3}$$

Equation 3 – Combine operation for a 4-byte word.

This requires four 4-input 'and' gates, one 4-input 'or' gate and six D-type Flip-flops.

In terms of overall complexity, the move from a standard byte-wide Aho-Corasick multi-string matching system to the technique described here requires us to replace a single FSM with w instances of a new FSM for matching sub-strings and one instance of the combine operation described above. The new FSM will have a similar number of states to the original, but will require a factor of w increase in the number of match outputs. Actual resource utilisation will depend on many parameters relating to the FSM implementation as will be shown later. The resources required for the combine operation are trivial for small values of w – but will grow rapidly in size with w as it implements a w^2 input Boolean function.

Implementation

Each FSM has to be able to match multiple substrings, and this can be done using the Aho-Corasick multiple string matching algorithm. As we are using a multiple string matching algorithm we can actually use each FSM to search for the substrings for several different search strings.

The method used here for the FSM implementation is table based – the reason for taking this approach is that we are able to have a fixed core of logic for any FSM (of a given size) and we determine the operation performed by the FSM by specifying the contents of the FSM table. The state transition table for such a FSM is very redundant, and this can be implemented using the type of FSM implementation described by (Sugawara et.al., 2004), as shown in Figure 3.

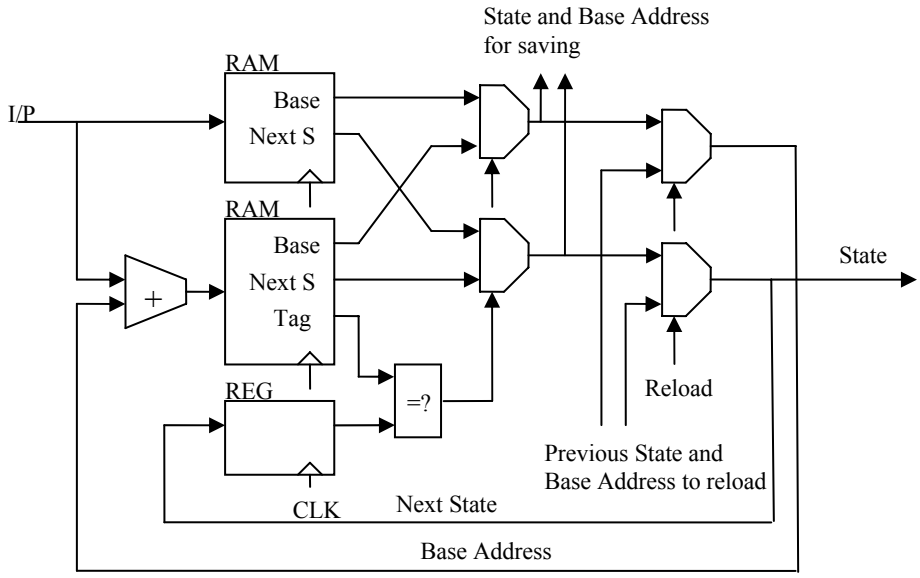


Figure 3 - Finite State Machine Implementation from (Sugawara et.al., 2004).

The algorithm by (Sugawara et.al., 2004) works on the basis that for a given input value i, a large proportion of transition table entries for current state s will be the same as for the IDLE (or initial) state. The algorithm uses a default table that contains table entries for all input values of i in the IDLE state. All we need in addition to this are the entries from the full state transition table that differ to the entries in the default table – this difference table is typically very sparse.

Table 2 gives an example of a simple FSM that searches for the single string "abcabc". The input is a numerical value that relates to the character shown in brackets. The state represents the portion of the search string that has been matched. The tables contain the next state for the FSM and in this example the match succeeds when the FSM is in state 6.

Original State Transition Table

Current State

Input	'' 0	'a' 1	'ab' 2	'abc' 3	'abca' 4	'abcab' 5	'abcabc' 6
0(a)	1	1	1	4	1	1	1
1(b)	0	2	0	0	5	0	0
2(c)	0	0	3	0	0	6	0
3(x)	0	0	0	0	0	0	0

Default Table

Input	
0(a)	1
1(b)	0
2(c)	0
3(x)	0

Difference Table

Current State

Input	'' 0	'a' 1	'ab' 2	'abc' 3	'abca' 4	'abcab' 5	'abcabc' 6
0(a)				4			
1(b)		2			5		
2(c)			3			6	
3(x)							

Table 2 – Simple FSM to match search string 'abcabc'.

To find the next state for any current state and input, we first look in the difference table. If this does not have an entry then we use the value from the default table instead.

This difference table is decomposed into a series of state vectors, and these are packed together (overlapping) into a one-dimensional packed array – carefully avoiding any collisions between active entries. Each entry in the packed array is tagged with the current state it belongs to. To retrieve an entry from the packed array we need to know the base address of the state vector for the current state in the packed array and then use the current input as an offset from that point. If the entry fetched from the array has a tag that is equal to the current state, then we have found a valid difference table entry – if not, there is no entry for the current state and input in the packed array, so we use the value from the default array for the current input.

We can see an example in Table 3 of the how the FSM in Table 2 is converted into this format. To improve performance, each entry (in both arrays) also contains the base address of the state vector in the packed array for the next state (NS).

State Vectors

0	1	2	3	4	5	6
			4			
	2					
		3				
				5		
					6	

Packed Array

NS	Base	Tag
4	2	3
2	0	1
3	0	2
5	2	4
6	0	5
-	-	-1

Default Array

NS	Base
1	0
0	0
0	0
0	0

0	0	0	0	2	2	0

: Base Addresses for state vectors

Table 3 – Packed and Default arrays, using the method described by (Sugawara et.al., 2004).

The algorithm by (Sugawara et.al., 2004) gives a significant memory saving for large FSMs, as we avoid the use of the two dimensional arrays. It is difficult to a give a figure for the resource utilisation based on string length or number of strings, as the memory required will be determined by how well the state vectors can be fitted together into the packed array.

Modifications to the FSM implementation

For our work, the design in Figure 3 was setup to operate on one packet at a time, by providing a restore input corresponding to the IDLE state. After testing the FSM design, it was found that performance was limited by the adder carry chain used in the implementation of the "+" operation that provides the index into the packed array. On investigation, it was found that it was possible to replace the "+" operator with a "bitwise XOR" operation. This is not as efficient as it will constrain any state vector to be within a single 2^n sized block of memory for a data input bus size of n-bits – in practice however it is not found to cause much reduction in efficiency, as seen later. The advantage is that it makes each bit in the address calculation independent which improves the hardware performance.

The state from each FSM is fed into a state decoder table that generates a substring match vector specifying all of the substrings that complete matching in the current state. The use of a table for this operation avoids needing to build specific logic for each string set. As strings shorter than the word size can generate a match from a subset of the FSMs, we allow one or more of its substrings to be the 'null string' which will match in any state, thus we always require a match from all FSMs irrespective of search string length.

Rule processing

Rather than generating a specific piece of hardware for a given rule set, it was decided that we should identify an efficient size of 'string matching engine' and then instantiate a number of these to cover the set of strings. We will not know in advance how many strings will fit into a FSM of any particular size, as this will depend on how compact the packed array can be made. The best size of FSM will depend on a number of factors, but will relate in particular to the memory resources available in the hardware. As we don't know in advance how many strings we can fit into a given FSM, we need to take an iterative approach and try increasing numbers of strings to see how many will actually fit.

Rule sets such as those defined by snort will allow us to have content matching that is case independent. We can deal with this by allocating these strings to separate 'string matching engines' to the ones used for strings that are case dependent and pre-pending an input function that maps all upper case letter to lower case.

Software

Software was written to take a set of strings and to build an Aho-Corasick trie for performing the matching. The design was optimised using standard techniques to enable the matching to be performed at a rate of one byte per clock cycle. From this, a state transition table was produced and then compressed using the technique described by (Sugawara et.al., 2004) and outlined above.

The first stage was to choose a sensible size for the FSM. The software was modified so that instead of reading in all the search strings, it stops after a certain limit of search characters had been exceeded and the memory resources required for that amount of search characters reported. This was repeated for a range of maximum numbers of search characters – the search strings being taken from a randomised order set of case dependant rules from the hogwash (Larsen & Haile, 2001) "insane" rule set. The operator for the packed array index was chosen as the ADD operator.

The tests were performed for a range of input bus sizes, and the memory requirements for a single 8-bit slice are given in Figure 4. We see that the amount of memory required increases with the input bus width, as the number of substrings increases with the number of 8-bit slices.

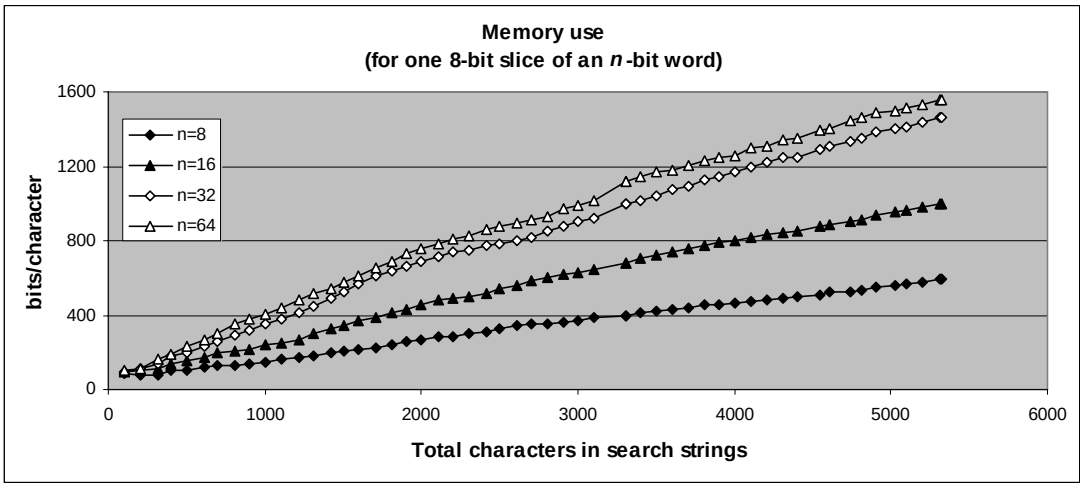


Figure 4 - Memory use.

The traces in Figure 4 are roughly linear for a range of total characters from 200 to 2000. The approximate memory requirements B_w in bits for word size w and character count s in this range are shown in Equation 4. Memory usage will of course vary with the particular set of strings chosen.

$$B_8 = 0.10s + 51, \quad B_{16} = 0.19s + 53, \quad B_{32} = 0.33s + 33, \quad B_{64} = 0.35s + 54$$

Equation 4 – Approximate memory use for a range of 200 to 2000 search characters.

When we get to a word size of 64-bits, the total amount of memory required does not increase as much as expected, as there are an increasing number of short identical substrings, including null strings. Calculations here show the exact amount of memory required – in practice the memory will only be available in particular sizes, as shown later.

Interestingly, the amount of memory required per search character increases with the total amount of characters in the search strings. This is partly due to the memory requirements of the state decoder, but this effect is present even if we don't take this into account. We would expect to get some gain as we increase the number of search strings as we should have nodes within the trie shared between multiple search strings. We can see this effect in Figure 5.

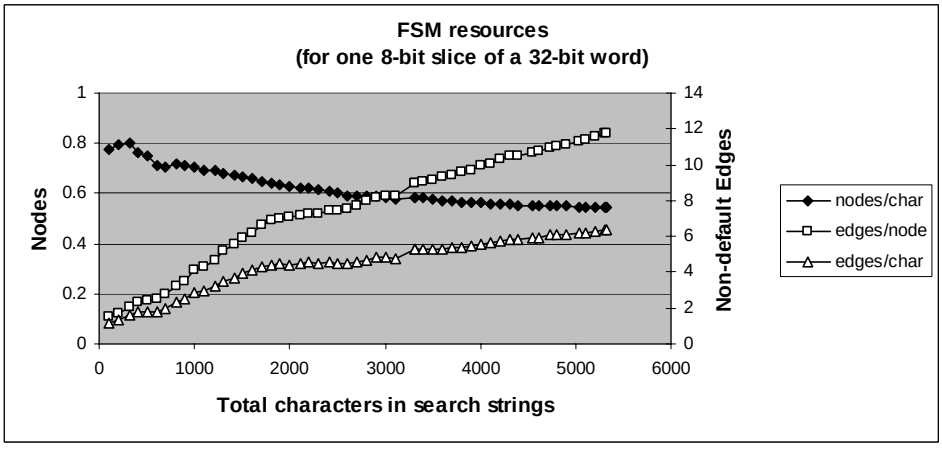


Figure 5 - FSM resources.

The number of trie nodes per search character decreases with the total number of search characters as expected; however this effect is counteracted by the increase in the number of 'non-default edges'. The 'non-default edges' are transitions from one node to another that are recorded in the packed array and this relates to the FSM becoming more complex and there being more interconnectivity between nodes. The overall effect is that the number of non-default edges per search character increases with the number of characters. The total memory requirements for the packed array are roughly proportional to the number of non-default edges, as each 'non-default edge' will require its own entry in the packed array.

All three traces in Figure 5 are roughly linear up to a total of 1700 characters in the search strings. The approximate results in this range, for a total of s characters are shown in Equation 5.

$$nodes/char = 0.79 - 9\times10^{-5}s$$

$$edges/node = 3.3\times10^{-3}s + 0.82$$

$$edges/char = 2.0\times10^{-3}s + 0.81$$

Equation 5 - Approximate resource usage for up to 1700 search characters.

From Figure 5, it can be seen that when using this style of implementation it is not necessarily the best option to have large Aho-Corasick FSMs. The resources used appear to be lower when only a small number of strings are searched for; this however will be dependent on the sizes of memory available for the various FSM tables.

Determining an optimal FSM size

The software was modified to re-run a number of tests for a fixed input word size of 32-bit, using variations of the algorithms. The tests were run for an increasing number of search strings and the total memory resources required were calculated for implementation within a Xilinx Virtex-II FPGA ("Xilinx Virtex-II", 2005) – this type of FPGA contains 18Kbit Block RAM primitives (BRAMs). As an experiment, we also test the effect of preceding each FSM input with a custom built compression table to reduce the redundancy in the input data – as shown in Figure 6.

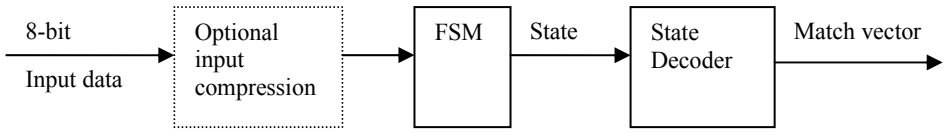


Figure 6 - One 8-bit slice of matching system.

The tests were run with either raw or compressed input data and with either ADD or XOR used for packed array indexing. The results are shown in the left graph of Figure 7; the four traces are very close together, and an enlarged section is given (for clarity) in the right graph where the resource utilisation is the lowest.

The use of compression did not have much effect when we use a large number of strings, however with a small number of strings the memory used increased because the extra memory needed for input compression was greater than the memory saved within the FSM tables. The choice of ADD or XOR algorithms had very little effect.

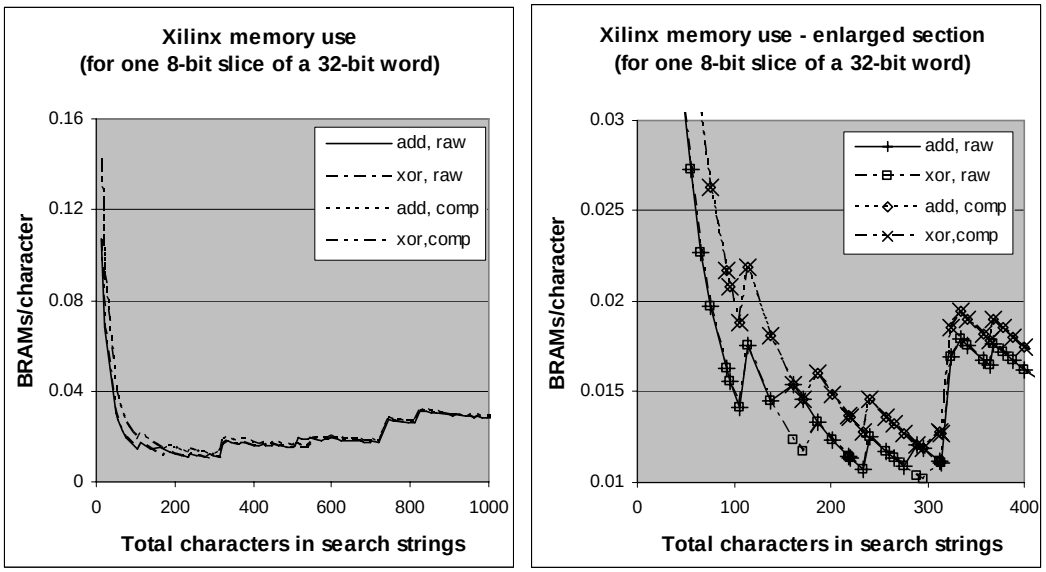


Figure 7 - Xilinx memory use vs. the number of search characters.

From the results shown in Figure 7, the best option appears to be a small FSM implementation dealing with a maximum of 200-300 search characters, and which uses 3 BRAM components. The best results for each of the four algorithm combinations are shown in Table 4.

FSM Input	Packing Algorithm	Search Strings	Search Characters
Raw	ADD	22	275
Raw	XOR	24	295
Compressed	ADD	18	234
Compressed	XOR	18	234

Table 4 - Maximum number of search strings for a 3 BRAM implementation.

Hardware implementation

A VHDL model was built of a 32-bit string matching engine that consisted of four 8-bit wide matching 'slices' and a unit to combine together the results – as shown in Figure 8. On the basis of the results above, a decision was made not to use input compression and to use the XOR function for indexing into the packed array for the FSM. The VHDL model was tested by simulation, the design synthesised and built for a Xilinx XC2V250-6 FPGA to determine its performance and resource utilisation. The design was also simulated "post place and route" to test the resulting FPGA design.

The parameters of the FSM design were taken from the rule processing results of the previous section. Each FSM has an 8-bit input, an 8-bit state variable and a 108-bit substring match output. (Note: The value of 108 was chosen as it is a multiple of one of the BRAM memory widths, which is 36-bits.) Four instances of the FSMs were used with a fixed combine operation to generate a matching engine having a 32-bit data input and a 27-bit match output. This is capable of matching up to 27 search strings in parallel, depending on the length of the strings. The use of the match vector output enables us to indicate matches of multiple search strings occurring at the same time; this match vector output could be used to generate an indication of which strings occurred within a given input data packet (including the detection of multiple matches of different strings).

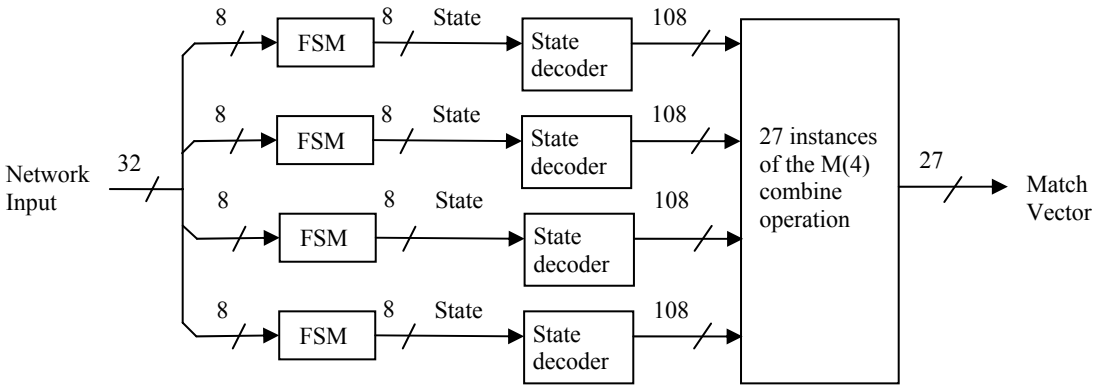


Figure 8 - Matching Engine.

Performance and resource utilisation

The VHDL model was first configured for using a bitwise XOR operator for the FSM table index operation and this gave a minimum clock cycle time of 6.7ns (149 MHz) – given the 32-bit input, this corresponds to a search rate of around 4.7 Gbps.

The resources required for a Xilinx XC2V250-6 FPGA were as follows:

- 12 Block RAM components (out of a total of 24)
- 250 logic slices (out of a total of 1536)

We can see from the above, that the size of any design will be limited by the Block RAM resources. The FPGA component used as the target of these experiments is however by current standards rather small. Taking the top of the range Virtex4FX FPGA as a comparison, we should be able to fit 46 of these matching engines within the FPGA, using all of the BRAM resources and around 20% of the logic slices. This should enable us to perform a parallel search of around 900 search strings.

For comparison, the VHDL model was rebuilt to use an ADD operator for the packed array indexing and this gave a minimum clock cycle time of 8.4 ns (119 MHz – giving a 3.6 Gbps search rate). This confirms the earlier assertion that using the bitwise XOR operator for the packed array indexing would be faster than using ADD.

Testing

The design has been tested by simulation with a large set of artificial input data containing various combinations of the strings being searched for, including: isolated instances of search strings; combinations of search strings at various spacing and overlap; and some strings rearranged in all 24 variations of the byte ordering in a group of 4 bytes. These tests were repeated for all four byte alignments of the input data – giving a total of 288 test cases. The results of the tests were compared with the expected outcomes to ensure that the search strings only matched as and when was expected. All tests passed correctly both for the original VHDL design and the post place and route simulations.

Pattern	Byte position that match occurs		
	"cybercop"	"gOrave"	"login: root"
`----cybercop==================`	12		
`----ycebcrpo==================`			
`----ybcecorp==================`			
`----cybercybercop==============`	16		
`----gOrave====================`		9	
`----login: root================`			14
`----logOrave==================`		11	
`----killogin: root=============`			17

Table 5 - A few of the test patterns used and the expected results.

A few examples of patterns used to test matching and the expected results are shown in Table 5.

Conclusion

This paper describes the design and simulation of a parallel algorithm for the implementation of high speed string matching; this uses fine-grained parallelism and performs matching of a search string by splitting the string into a set of interleaved substrings and then matching all of the substrings simultaneously.

We show that the FSM implementation technique described by (Sugawara et.al., 2004) can be modified by the use of bitwise XOR in place of ADD for the indexing operation to improve its performance. We also see that this implementation can be optimised in terms of resource utilisation by the choice of FSM size.

A VHDL model of a string matching engine based on the above ideas has been produced, synthesised and built for a Xilinx FPGA and tested via simulation. The results show a search rate of around 4.7 Gbps for a 32-bit input word. The design is table based and changes to the search strings can be made by generating new contents for the tables rather than having to generate a new logic design – this is particularly important for systems being updated in the field.

Future Work

One area where the resources in this design could be reduced is in the state decoder table – which accounts for 50% of the memory resources. This gives a substring match vector for the current state of the FSM – thus showing which substrings match in a given state. This table could be replaced with a piece of logic, but this would need to be rebuilt for every set of strings.

Further work is needed to see if the memory requirements for the state decoder can be decreased, possibly taking advantage of the redundancy that exists within this table. This could for example be replaced by a two stage decoder design. Finally it would also be interesting to see if any parts of the state decoder could be implemented as fixed logic.

References

Abbes, T., Bouhoula, A., & Rusinowitch, M. (2004). Protocol Analysis in Intrusion Detection Using Decision Tree. In proceedings of International Conference on Information Technology: Coding and Computing (ITCC'04), Volume 1 (pp. 404-408). Las Vegas, Nevada.

Aho, A.V., & Corasick, M.J. (1975). Efficient string matching: an aid to bibliographic search. Communications of the ACM, 18(6), 333-340.

Attig, M., & Lockwood, J.W. (2005). SIFT: Snort Intrusion Filter for TCP. In Proceedings of IEEE Symposium on High Performance Interconnects (Hot Interconnects-13). Stanford, California.

Baker, Z.K., & Prasanna, V.K. (2004). A methodology for Synthesis of Efficient Intrusion Detection Systems on FPGAs. In proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines FCCM '04. Napa, California.

Boyer, R.S., & Moore, J.S. (1977). A Fast String Searching Algorithm. Communications of the Association for Computing Machinery, 20(10), 762-772.

Cho, Y., & Mangione-Smith, W. (2004). Deep Packet Filter with Dedicated Logic and Read Only Memories. In proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines FCCM '04. Napa, California.

Clark, C., & Schimmel, D. (2004). Scalable Multi-Pattern Matching on High-Speed Networks. In proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines FCCM '04. Napa, California.

Fisk, M., & Varghese, G. (2001). An Analysis of Fast String Matching Applied to Content-Based Forwarding and Intrusion Detection (successor to UCSD TR CS2001-0670, UC San Diego, 2001). Retrieved 9 March 2006, from http://public.lanl.gov/mfisk/papers/setmatch-raid.pdf

Franklin, R., Carver, D., & Hutchings, B.L. (2002). Assisting Network Intrusion Detection with Reconfigurable Hardware. In proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines FCCM '02 (pp.111-120). Napa, California, USA.

Hopcroft, J.E., Motwani, R., & Ullman, J.D. (2001). Introduction to Automata Theory, Languages and Computation, 2nd Edition : Addison Wesley.

Iyer, S., Rao Kompella, R., Shelat, A. (2001). ClassiPi: An Architecture for fast and flexible Packet Classification. IEEE Network, 15(2), 33-41.

Knuth, D.E., Morris J.H., & Pratt, V.B. (1977). Fast pattern matching in strings. SIAM Journal of Computing, 6(2), 323-350.

Kruegel, C., & Toth, T. (2003). Using Decision Trees to Improve Signature-based Intrusion Detection. In Proceedings of the 6th Symposium on Recent Advances in Intrusion Detection (RAID2003), Lecture Notes in Computer Science, LNCS 2820 (pp. 173-191). Springer Verlag.

Larsen , J., & Haile, J. (2001). Securing an Unpatchable Webserver … HogWash. Retrieved 9 March 2006, from http://www.securityfocus.com/infocus/1208

Moscola, J., Lockwood, J., Loui, R.P., & Pachos, M. (2003). Implementation of a content-scanning module for an internet firewall. In proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines FCCM ’03. Napa, California.

Paul, O. (2004). Improving Distributed Firewalls Performance through Vertical Load Balancing. In proceedings of Third IFIP-TC6 Networking Conference, NETWORKING 2004, Lecture Notes in Computer Science, LNCS 3042 (pp. 25-37). Springer-Verlag.

Roesch, M. (1999). Snort - Lightweight Intrusion Detection for Networks. In proceedings of LISA '99: 13th Systems Administration Conference (pp. 229-238). Seattle, WA : USENIX.

Sidhu, R. & Prasanna, V.K. (2001). Fast Regular Expression Matching using FPGAs. In proceedings of the 9th International IEEE symposium on FPGAs for Custom Computing Machines, FCCM’01. Rohnert Park, California, USA.

Sugawara, Y., Inaba, M., & Hiraki, K. (2004). Over 10 Gbps String Matching Mechanism for Multi-stream Packet Scanning Systems. In proceedings of Field Programmable Logic and Applications, 14th International Conference, FPL 2004 (pp. 484-493). Springer-Verlag.

Tan, L., & Sherwood, T. (2005). A High Throughput String Matching Architecture for Intrusion Detection and Prevention. In the proceedings of the 32nd Annual International Symposium on Computer Architecture (ISCA 2005). Madison, Wisconsin, USA.

Tripp, G. (2005). A finite-state-machine based string matching system for intrusion detection on high-speed networks. In Paul Turner and Vlasti Broucek, editors, EICAR Conference Best Paper Proceedings, (pp. 26-40). Saint Julians, Malta : EICAR.

Xilinx Virtex-II Platform FPGAs: Complete Data Sheet – Product Specification. (2005). Xilinx Inc. Retrieved 9 March 2006 from http://direct.xilinx.com/bvdocs/publications/ds031.pdf

An Untraceability Protocol for Mobile Agents and Its Enhanced Security Study

Rafał Leszczyna[1] *and Janusz Górski*[2]

[1] *European Commission, Joint Research Centre,*
Via Enrico Fermi, 21020 Ispra (VA), Italy
[2] *Gdansk University of Technology,*
Narutowicza 11/12, Gdańsk, Poland

About Authors

Rafał Leszczyna, is a PhD student at Gdansk University of Technology, Poland. He graduated in 2002 with an MSc in Distributed Computer Systems. In 2004 he joined the Cybersecurity group at the European Commission Joint Research Centre. He is also a member of the Information Assurance Group led by professor Górski. Currently he is finishing his PhD thesis 'Architecture supporting security of agent systems' which summarises the overall work conducted in the last four years. His research focuses on security of computer systems, security protocols and software agents.

Janusz Górski is a professor and head of Department of Software Engineering at Gdansk University of Technology. His research interest is in risk management, and trust, safety and security of IT systems and infrastructures. He led numerous international and national research projects related to these domains. Presently he is leading the Information Assurance research group focusing on trust and security assurance and risk management. He provides consultancy to industrial partners and national and international agencies. He acted as the organisation and programme chair of several national and international conferences related to software engineering and safety and security of software based systems.

Keywords

Wireless Security, ICT Security, Privacy, Anonymity, Untraceability, Security Protocols, Security Analysis, Cryptographic Protocol Verification, Software Agents, Mobile Agents

An Untraceability Protocol for Mobile Agents and Its Enhanced Security Study

Abstract

Among various computational models to accommodate the mobile wireless computing paradigm, the mobile agent model has major technological advantages. Moreover, our research shows that the model has some unique features which, if used appropriately, may support user privacy. We therefore propose a security protocol for mobile agents aiming at user untraceability. We have performed a basic security analysis of the protocol and evaluated its performance. We have also implemented it, made it available to users, and applied it to an e-health service. In this paper, we present an enhanced security study of the protocol. As far as we know, such systematised study has not yet been performed. Additionally, known analyses limit their interest to particular attackers (internal attackers). We believe that the attack checklist which we developed to evaluate our protocol, may serve as a helpful reference for authors of other protocols. The paper is also addressed to designers and administrators of mobile agent systems who consider provision of user untraceability within their systems.

1 Introduction

To accommodate the new computing paradigm introduced by mobile wireless computing, various software models have been proposed including the client/agent/server, the client/intercept, the peer-to-peer, and the Mobile Agent (MA) models (Spyrou, Samaras, Pitoura, & Evripidou, 2004).

Among them the MA model appears to be particularly suitable due to its technological advantages including bandwidth conservation, latency reduction, disconnected operation, load balancing and dynamic deployment (Gray, Kotz, Cybenko, & Rus, 2000; Chess et al., 1995). These features correspond to characteristics of mobile wireless networks, which, compared with traditional fixed networks, present lower transfers, less reliable and more expensive communications, lower computational power, more lightweight devices and flexible locations of participating devices (Spyrou et al., 2004).

Furthermore, (Spyrou et al., 2004) presents a MA framework able to instantiate dynamically each of the software models. The authors claim that their MA implementation of the models further enhances their applicability to mobile wireless computing, making applications more light-weight, tolerant to intermittent connectivity and adaptable to dynamically changing environments.

The primary focus of our research into MA, has been methods for their protection, since security is one of the biggest challenges in the field (Jansen & Karygiannis, 2000). In the progress of this research, we discovered that Mobile Agent Systems (MAS) have some characteristics which may facilitate protection of users' privacy (Leszczyna & Górski, 2004). Based on the observation we proposed a security protocol aiming at anonymity (precisely – untraceability) of agent users (Leszczyna & Górski, 2005b, 2004). We evaluated its performance (Leszczyna & Górski, 2005a) and performed a basic security study (Leszczyna & Górski, 2004). We implemented the protocol (Leszczyna, 2005b) as an open source package (the package is available at `http://jade.tilab.`

`com/community-3rdpartysw.htm#Untraceability`) and deployed it in an e-Health anonymous counselling scenario (Leszczyna, 2005a).

In this paper we present an extended security study of the protocol.

To prepare this study we applied known Traffic Analysis (TA) attacks to MAS and discussed their feasibility. This way a list of attacks against untraceability protocols for MA was created, against which we then validated our protocol.

The study shows that certain TA attacks are very potent against untraceability protocols. For them we present additional measures which can be introduced to increase the level of security.

As far as we know, such systematised study has not been performed until now since previous analyses limit their interest to particular attackers (internal attackers). We note that the attack checklist may also serve as a helpful reference for authors of another protocols. The part focusing on additional measures in case of effective attacks is addressed to designers and administrators of MAS who might consider employing an untraceability protocol.

In the next subsections we present a notion of anonymity and introduce a minimal set of fundamental requirements for software agents required to understand the paper. Next we briefly describe the protocol and finally move on to the security analysis.

1.1 Anonymity

Anonymity is the property that ensures that a user may use a resource or a service (the *items of interest*) without disclosing their identity (National Institute of Standards and Technology (NIST), 1998). *Untraceability* is a subclass of communication anonymity (Pfitzmann & Hansen, 2005) assuring that the identity can not be inferred by tracing a message (an agent – in case of MAS).

Anonymity plays a crucial role in many activities conducted in the Internet. For example users may be reluctant to engage in web-browsing, message-sending and file-sharing, unless they receive guarantees that their anonymity will be protected to some reasonable degree (Halpern & O'Neill, 2004). (Gülcü & Tsudik, 1996) describe four categories of internet applications where anonymity is required[1]: discussion of sensitive and personal issues, information searches, freedom of speech in intolerant environments and polling/surveying.

1.2 Mobile (Autonomous) Agents

Since no established taxonomy of agents exists, we support the notion of generic *software agents* being software programs acting on behalf of users (this is an etymologically inspired definition, see (Harper, 2006; Merriam-Webster Incorporated, 2006)). Then we use adjectives to describe a particular type of agent which is acting in our interest (after the (Franklin & Graesser, 1996) and (Murch & Johnson, 1999) classifications). In this article we are in principle concentrated on mobile autonomous agents i.e. those able to exercise control over their own actions (*autonomous*) and able to roam networks freely (*mobile*).

[1]It is important to note that the authors don't claim this set to be exhaustive.

Generally an agent's structure is described in terms of a *state* and a *behaviour*. Roughly speaking, the term behaviour refers to an agent's code and the state to the data held by the agent.

Mobile agents are given a *goal* and to accomplish it they roam from one network node (in MAS, called a *container*) to another, starting from a *base*. The sequence of containers to be passed is called a *route*. The agents' goal can be formulated through explicit indication of a node to be visited or in a more abstract way, without indicating nodes, for instance: to provide information about the cheapest vendor of a certain product. In the former case the node is called a *destination*. Note that in the case of 'abstract' goal formulation, the identity of receiver is naturally obscured.

1.3 The Protocol

In common scenarios, agents cover a route and come back to their base. To allow them to return, the identifier of the base must be explicitly stored in their state. Our protocol aims at obscuring this identifier.

Very few proposals have been made thus far for MA untraceability (Goldschlag, Reed, & Syverson, 1996; Westhoff, Markus Schneider, Unger, & Kaderali, 2000; Enzmann, Kunz, & Schneider, 2002) and these are based on conception of onion routing (Reed, Syverson, & Goldschlag, 1998), which requires a route determination before launching an agent. Unfortunately such an approach restricts spontaneous route selection during migration by an agent, an advantage over traditional message-based communication, which is required for certain applications (targeting dynamic deployment, load balancing etc (Gray et al., 2000)). Our protocol doesn't introduce this limitation.

In brief, the concept of the protocol is that at each visited container an agent asks the container to encrypt the identifier of previous container using the current container's secret key. Then the encrypted identifier is put into the LIFO queue stored in the agent's state. After achieving its goal, when the agent wishes to come back to its base container, it uses the queue to find its way back. Down the route back, the identifiers are subsequently decrypted by containers using their secret keys. For a more formal specification of the protocol, refer to our earlier work (Leszczyna & Górski, 2005b).

2 Security Study

Cryptographic Protocol Verification (CPV) methods spam into two groups: formal and informal.

The informal methods such as evaluation based on a list of known attacks (which we discuss later in this section) or attack-oriented testing (testers try to break the security of protocols), are clear and straightforward, but they don't give full warranties of security. What they say is that a protocol is safe from *certain* attacks. However a new attack can always appear, to which the protocol will be prone.

For this reason some people consider using formal methods more appropriate. Formal methods for CPV (a nice overview of which can be found in works of Meadows, e.g. (Meadows, 2003a, 2003b)) are based on a model of the protocol and the context of its use (usually: its users (principals); a communication protocol (communication channels and messages exchanged), an attacker

and security measures) and its verification. Formal methods aim at providing full mathematical proofs of correctness and to achieve this goal mathematical constructs are employed such as logics and algebras. Unfortunately, in practice formal models of protocol contexts are often too complex to allow their verification (and often also their creation) in a reasonable timescale, thus some simplifying assumptions are usually taken. In consequence warranties given by formal approaches also tend to be limited.

Apart from a few fields where particular approaches tend to fit extraordinarily well (e.g. formal methods seem to work well with authentication protocols) the most common opinion is that there is no silver bullet and different complementary methods should be used.

In the case of untraceability protocols the context is so complex that they are particularly difficult to approach formally (Meadows, 2003a). Attacker modelling seems to be an especially demanding task. The adversary can compromise communication media (*external*) and/or mix nodes, recipients and senders (*internal*) (Raymond, 2001). The adversary may succeed in attacking all nodes (*omnipresent*) or `k` of them (*k-listening* (Dolev & Ostrobsky, 2000)) *actively* or *passively*. An active adversary can arbitrarily modify the computations and messages (by adding and deleting) whereas a passive adversary can only listen (Raymond, 2001). Finally different configurations, hybrids and alliances of attackers may occur, such as external-active or colluding internal and external.

Such an extensive model of an attacker means that so far most security analysis of untraceability protocols is based on a list of known attacks, which is performed through taking each known attack and referring it to the proposed protocol. Additionally some evaluators deliver probabilistic assessments which aim at providing a wider picture of a protocol's security (Kesdogan, Egner, & Büschkes, 1998; Goldberg & Wagner, 1998; Reiter & Rubin, 1998).

Some formal frameworks were proposed for untraceability protocols (e.g. (Syverson & Stubblebine, 1999; Garcia, Hasuo, Pieters, & Rossum, 2005)), however they are not supported by automatic checkers. This imposes a need for manual validation and makes verification a demanding task (for a notation must be learned). Also an ordinary user may feel quite bewildered when presented such a mathematical proof method.

As far as verification of untraceability protocols for mobile agents is concerned, until now evaluation based on a list of attacks has been performed, but this has been limited to models of internal attackers, which we consider as insufficient. We also conducted such a study, which helped to expose an attack and motivated us to present measures to harden the protocol against resistant it[2], but this had to be followed by further studies taking into account other types of attackers.

In this section we present the results of such extended evaluation. We intend to accompany this with more formal analyses in the future.

The analysis shows that an attacker aiming at compromising the anonymity of agent owners

[2]Not without a price. Because the attack is specific, requiring a particularly strong attacker, so it needs correspondingly, strong (and costly) countermeasures. Thus we decided to propose an alternative protocol (Leszczyna & Górski, 2004; Leszczyna, 2005a).

has available such a large variety of attack paths that it is impossible to defend it using only an untraceability protocol.

Other means (e.g. dummy traffic, agent delaying) must be introduced into the system in which communication anonymity is to be preserved (which will result in an *untraceability architecture*). In this paper, when discussing particular attacks, we recall these countermeasures. We decided not to include the measures in the protocol specification, giving freedom to designers and administrators to decide whether to increase the level of anonymity at the cost of efficiency.

Because both attacks and countermeasures are also relevant to other untraceability protocols for software agents(Goldschlag et al., 1996; Westhoff et al., 2000; Enzmann et al., 2002), the study, which originally was dedicated only to our protocol, in the end gives more general conclusions applicable to other protocols.

To construct the list of attacks against untraceability protocols for MAS we reviewed anonymity bibliography available at `www.freehaven.net/anonbib/`. Of the prominent studies we reviewed (e.g. (Kesdogan et al., 1998; Goldberg & Wagner, 1998; Reiter & Rubin, 1998; Mazières & Kaashoek, 1998; Dolev & Ostrobsky, 2000; Raymond, 2001)), Raymond's appears to be the most comprehensive (it actually summarises all the others). Thus we took Raymond's work and discussed all the attacks it describes step by step referring them to MAS.

Note that different attacks assume different types of adversary, so we don't state a general attacker model. It should rather be considered on a case by case basis depending on the attack.

2.1 Attacks Based on the Agent's Distinguishing Features

If an external attacker is able to read an agent state (which usually is the case), they can easily recognise and trace the agent through subsequent containers, as long as the state is easy distinguishable from the states of other agents. Thus, in such cases, obfuscation of the agent state is required. This is performed in such a way that the state is different before and after traversing each container. Because to provide such means of state obfuscation is normally out of the scope of untraceability protocols for MA, it is up to designers and administrators to consider various cryptographic schemas for this obfuscation e.g. encryption using a public key of the next station.

2.2 Brute Force Attack

If the obfuscation is introduced, a powerful (either omnipresent or able to move to all places in the network) external attacker can follow every possible route the agent could have passed through. They observe the agent entering a container and then follow all outgoing agents. They can do so for each of the agents on and on until the destination is reached.

If it can be guaranteed that containers are always visited by a certain number of agents and more than one leaves a container at the same moment, then the probability of linking a base and a destination decreases exponentially with the length of the route. However it may happen that a network has low traffic and at a particular instant, the agent was the only one which traversed a given route. Then it is totally exposed to an adversary.

To avoid this, in traditional networks, dummy traffic is usually introduced. This is a costly option, overloading the network, narrowing its bandwidth etc. But in the case of MAS, it is more worthwhile. In MAS all nodes are equal in the sense that there is no distinction between mixes and ordinary nodes, yet all nodes are mixes. So the attacker cannot easily detect which node is the destination one. They might guess that it is the last one before returning to the base, but it is enough to force the agent to traverse additional containers on its way back to prevent such inferences.

To hide the base station, one option is to introduce 'redundant' agents which start their lifecycle on random containers and then roam around the network waiting for a task. Thus when a user needs an agent they pick it up rather than launch. Because the agent started its lifecycle earlier on a container different to the one where it is being picked up, this previous container is the base container and not the one where the agent was employed.

This means of protection is not as strong as the introduction of dummy traffic, because the probability decreases linearly with the route length, but it is much less costly and may be sufficient, especially if the network is naturally loaded with normal agent traffic.

2.3 Timing Attacks

If traversing a route of a particular length always takes an agent a specific time, then an external adversary may correlate times taken by observed agents. Raymond, for example (Raymond, 2001) assumes two routes to cover, for which subsequently 1 second and 3 seconds are always required. Then if two messages are observed arriving in the network at 0:00 and 0:01 and leaving it at 0:03 and 0:02, discovering which message passed which route is trivial.

In MAS, timing attacks are much less successful due to the difficulty in identifying the destination (see section 2.2). To increase this already high level of anonymity it is worth considering the introduction of random delays of agents visits to containers or batch processing (see the next section).

Note, that in practical agent environments, in which agents roam and perform tasks, random delays may naturally result from the tasks.

2.4 The Node Flushing Attack (a.k.a. Spam Attack, Flooding Attack, n-1 Attack)

In traditional networks a popular method of protection against TA is batch processing – a mix waits until `n` messages are collected before flushing them. This method protects from timing attacks. This is done at the price of message delays, which in case of reasonable network traffic may be acceptable if we take into consideration the quality received in return.

One attacker response is the *n-1 attack* in which they 'fill' the mix with `n-1` of their messages, so the mix will be left with `n-1` messages which are well distinguishable to the adversary and separated from message of interest. Thus we must assure that recognising their messages (agents) is not an easy task for an adversary. For this, we recommend to use the obfuscation method mentioned in section 2.1 together with nonces.

2.5 Contextual Attacks

These are attacks related to communication habits of users. Raymond distinguishes between three general types of attacks in this group: Communication Pattern Attacks, Packet Counting Attacks and Intersection Attack (Raymond, 2001).

All these attacks are applicable to MAS and, as with the traditional networks – they are difficult to protect from. This is mainly because they rely on actors being out of the control of system designers and administrators.

Communication Pattern Attacks – refers directly to the users' habits such as time of using network services and thus the user must not to have habits which are too unusual, if they want to remain anonymous.

Packet Counting Attacks – in this case, a user makes the task of tracing them easier by launching an agent of a characteristic, distinctive size (in terms of number of packets). To avoid this, a standard size of agent may be used, but this leads either to costly redundancies if the size is large, or to inconvenient boundaries, if the size is too small, imposing the distribution of data between more than one agent. Generally, protection from packet counting attacks seems to be hardly achievable using systemic methods.

Intersection Attacks – where an adversary observes network traffic and stepwise narrows the range of possible interlocutors (as described in (Berthold, Pfitzmann, & Standtke, 2000)). Such attacks actually undermine the defence of using *different* routes each time an agent goes to the same destination. Imagine an agent travelling twice from a base `A` to a destination `B`, each time passing through completely different containers. If an adversary observes these two trips, they notice that the only containers in common are `A` and the `B`, which makes them good candidates for interlocutors.

In conclusion, counteracting contextual attacks must be the user's responsibility: users should be conscious of the fact that by performing characteristic activities, they become more distinct and thus easier to trace.

2.6 Denial of Service Attacks

An attacker compromises some mix-nodes (making them inoperative), counting on fact that it will force a user to send their messages through them to change their behaviour. Though this attack works in the case of conventional networks, when applied to MAS, it appears to be ineffective, since agents arbitrarily choose a different route each time they roam. In case some containers are nonfunctional, they simply omit them and choose other. We just have to make sure that an agent will not behave abnormally when the destination container is compromised. It means that the agent should perform the whole route as if nothing had happened (not quickly return to its origin to report the damage).

2.7 Active Attacks Exploiting User Reactions

An example of such an attack is message interception and broadcasting to all possible recipients. The idea is that the original recipient will behave differently from the others. To prevent this kind of attacks, as in case of DoS attacks (see section 2.6), the feature of completing the whole route even in unusual situations (this time it is that the declared destination appears not to be the real one) should be included.

2.8 Agent Delaying

In this case, an external attacker stops an agent until he gathers enough network resources or until the network becomes easier to monitor or to see if possible recipients receive other messages, etc. To protect from this attack, administrators should consider introducing authenticated timing information aiming at detection of unwanted delays.

2.9 Agent Tagging

An adversary marks an agent, to facilitate its tracing.

Feasibility of attacks where distinct data are used for marking is very dependant on a particular system's architecture and should be discussed in relation to a real environment. In most cases, because of network protocol characteristics, such tagging is not feasible on the network layer, what makes the attack unavailable to an external attacker.

An internal attacker may perform it, however because of the above (see section 2.2) mentioned similarity between mixes and ordinary stations in MAS, it becomes ineffective.

Other known tagging attacks are *Agent Delaying* and *Shadowing*.

Agent Delaying – in contrast to (see section 2.8), this attack aims at distinguishing an agent in the network. This can be achieved through ensuring that the agent stops at each container for a specific, characteristic (but not suspect) time. This attack is difficult to prevent, but fortunately, also to perform, since it requires an external omnipresent attacker with the ability to modify the agent behaviour.

Shadowing – an attacker intercepts messages and copies them. After this, `k` copies of the message traverse the same route. Effective in traditional networks, in MAS (and using our 'autonomy-friendly' protocol) it appears to be useless since each of the copies autonomously chooses its own route. Thus it is important to make sure that agents choose their routes in a nondeterministic way. Deterministic routing forms a serious vulnerability – if adversaries can discover its algorithm they could predict agent routes.

2.10 Corrupted Party Attacks

Taking over the sender does not lead to any gain in knowledge because of the autonomous routing feature sustained by our protocol.

The interesting case is when the destination party is corrupted. According to (Raymond, 2001), the attack assumes that an adversary is able to encode some indicative information into a message (an agent – in our case) thus making it distinct to others. Raymond accompanies his description with two examples: varying the reply latency and sending messages of a specific length. With this description the attack appears to be a tagging attack performed by an active internal adversary, which we have already discussed in section 2.9.

3 Conclusions

We performed the enhanced security study, which in our opinion extends the analyses provided so far for untraceability protocols of MA.

To perform the study we discussed known TA attacks for traditional networks and, based on this discussion, prepared a list of TA attacks against MA. The list may be a helpful reference in security studies of other protocols.

The analysis shows that some features of MAS support anonymity-friendliness. Primarily we mean the difficulty in identifying the base station, which stems from the fact that all containers are potential mixes. Also autonomous routing, which, in contrast with other known untraceability protocols for MA, is preserved in ours, helps avoid many attacks.

On the other hand, we indicate points where a powerful enough (e.g. external) attacker may be successful in their attempt to trace an agent. Here, we recall that the decision of whether to introduce countermeasures or not should be left to system designers and administrators.

References

Berthold, O., Pfitzmann, A., & Standtke, R. (2000, July). The disadvantages of free MIX routes and how to overcome them. In H. Federrath (Ed.), *Proceedings of designing privacy enhancing technologies: Workshop on design issues in anonymity and unobservability* (pp. 30–45). Springer-Verlag, LNCS 2009. (`http://www.tik.ee.ethz.ch/~weiler/lehre/netsec/Unterlagen/anon/disadvantages_berthold.pdf`)

Chess, D., Grosof, B., Harrison, C., Levine, D., Parris, C., & Tsudik, G. (1995). Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, *2*(5), 34–49. (`citeseer.ist.psu.edu/article/chess95itinerant.html`)

Dolev, S., & Ostrobsky, R. (2000). Xor-trees for efficient anonymous multicast and reception. *ACM Trans. Inf. Syst. Secur.*, *3*(2), 63–84. (`www.cs.ucla.edu/~rafail/PUBLIC/31.ps`)

Enzmann, M., Kunz, T., & Schneider, M. (2002, July). Using mobile agents for privacy amplification in the trade with tangible goods. In *6th world multi-conference on systemics, cybernetics and informatics (sci)* (Vol. IV). Orlando, Florida, USA.

Franklin, S., & Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent agents III. agent theories, architectures and languages (ATAL'96)* (Vol. 1193). Berlin, Germany: Springer-Verlag.

Garcia, F. D., Hasuo, I., Pieters, W., & Rossum, P. van. (2005). Provable anonymity. In *Fmse '05: Proceedings of the 2005 acm workshop on formal methods in security engineering* (pp. 63–72).

New York, NY, USA: ACM Press. (http://doi.acm.org/10.1145/1103576.1103585)

Goldberg, I., & Wagner, D. (1998, August). TAZ servers and the rewebber network: Enabling anonymous publishing on the world wide web. *First Monday*, *3*(4). (http://www.ovmj.org/GNUnet/papers/goldberg97taz.ps)

Goldschlag, D. M., Reed, M. G., & Syverson, P. F. (1996, May). Hiding Routing Information. In R. Anderson (Ed.), *Proceedings of information hiding: First international workshop* (pp. 137–150). Springer-Verlag, LNCS 1174.

Gray, R. S., Kotz, D., Cybenko, G., & Rus, D. (2000). *Mobile agents: Motivations and state-of-the-art systems* (Tech. Rep. No. TR2000-365). Hanover, NH: Dartmouth College.

Gülcü, C., & Tsudik, G. (1996). Mixing email with Babel. In *Proceedings of the 1996 symposium on network and distributed system security (sndss '96)* (p. 2). IEEE Computer Society.

Halpern, J. Y., & O'Neill, K. R. (2004). Anonymity and information hiding in multiagent systems. *Journal of Computer Security*. (http://www.cs.cornell.edu/people/oneill/papers/jcs_halpern_oneill.pdf)

Harper, D. (2006). *Online etymology dictionary.* Website. (http://www.etymonline.com/ (last access: May 4, 2005))

Jansen, W., & Karygiannis, T. (2000). *Nist special publication 800-19 - mobile agent security.* (citeseer.ist.psu.edu/jansen00nist.html)

Kesdogan, D., Egner, J., & Büschkes, R. (1998). Stop-and-go MIXes: Providing probabilistic anonymity in an open system. In *Proceedings of information hiding workshop (ih 1998).* Springer-Verlag, LNCS 1525.

Leszczyna, R. (2005a, September). The solution for anonymous access of it services and its application to e-health counselling. In *Proceedings of the 1st 2005 ieee international conference on technologies for homeland security and safety (tehoss '05)* (Vol. 1, pp. 161–170). Gdańsk, Poland: Gdańsk University of Technology.

Leszczyna, R. (2005b, September). *Untraceability i add-on for jade.* Via Enrico Fermi 1, Ispra, Italy. (The add-on is available to download at http://jade.tilab.com/community-3rdpartysw.htm#Untraceability (last access: December 16, 2005))

Leszczyna, R., & Górski, J. (2004, December). *Untraceability of mobile agents* (Tech. Rep.). European Commission, Joint Research Centre, Institute for the Protection and security of the Citizen.

Leszczyna, R., & Górski, J. (2005a, October). *Performance analysis of untraceability protocols for mobile agents using an adaptable framework.* (Accepted for 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '06), Future University-Hakodate, 8 – 12 May 2006)

Leszczyna, R., & Górski, J. (2005b, July). Untraceability of mobile agents. In *Proceedings of the 4th international joint conference on autonomous agents and multiagent systems (aamas '05)* (Vol. 3, p. 1233-1234). Utrecht, the Netherlands.

Mazières, D., & Kaashoek, M. F. (1998, November). The Design, Implementation and Operation of an Email Pseudonym Server. In *Proceedings of the 5th ACM Conference on Computer and Communications Security (CCS 1998).* ACM Press. (ftp://cag.lcs.mit.edu/pub/dm/papers/mazieres:pnym.pdf)

Meadows, C. A. (2003a, January). Formal methods for cryptographic protocol analysis: emerging

issues and trends. *Selected Areas in Communications, IEEE Journal on*, *21*(1), 44-54. (Formal methods for cryptographic protocol analysis: emerging issues and trends; Meadows, C.; Naval Res. Lab., Washington, DC; This paper appears in: Selected Areas in Communications, IEEE Journal on; Publication Date: Jan 2003; On page(s): 44- 54; Volume: 21, Issue: 1; ISSN: 0733-8716)

Meadows, C. A. (2003b). *Towards a hierarchy of cryptographic protocol models.*

Merriam-Webster Incorporated. (2006). *Merriam-Webster Online.* Website. (`http://www.m-w.com/` (last access: May 4, 2005))

Murch, R., & Johnson, T. (1999). *Intelligent software agents.* Prentice Hall PTR.

National Institute of Standards and Technology (NIST). (1998). *Common criteria for information technology security evaluation - part 2: Security funtional requirements.* U.S. Government Printing Office.

Pfitzmann, A., & Hansen, M. (2005, December). *Anonymity, unlinkability, unobservability, pseudonymity, and identity management a consolidated proposal for terminology (version v0.25).* Website. (`http://dud.inf.tu-dresden.de/Anon_Terminology.shtml` (last access: February 15, 2006))

Raymond, J.-F. (2001). Traffic analysis: Protocols, attacks, design issues and open problems. In H. Federrath (Ed.), *Designing privacy enhancing technologies: Proceedings of international workshop on design issues in anonymity and unobservability* (Vol. 2009, pp. 10–29). New York, NY, USA: Springer-Verlag New York, Inc. (`citeseer.ist.psu.edu/454354.html`)

Reed, M. G., Syverson, P. F., & Goldschlag, D. M. (1998). Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, *16*(4).

Reiter, M., & Rubin, A. (1998, June). Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, *1*(1). (`http://avirubin.com/crowds.pdf`)

Spyrou, C., Samaras, G., Pitoura, E., & Evripidou, P. (2004, October). Mobile agents for wireless computing: The convergence of wireless computational models with mobile-agent technologies. *Mob. Netw. Appl.*, *9*(5), 517–528. (`http://doi.acm.org/10.1145/1027347.1027354`)

Syverson, P. F., & Stubblebine, S. G. (1999, September). Group principals and the formalization of anonymity. In *Proceedings of the world congress on formal methods* (Vol. 1, pp. 314–333). Springer-Verlag, LNCS 1708. (`citeseer.ist.psu.edu/syverson99group.html`)

Westhoff, D., Markus Schneider, I., Unger, C., & Kaderali, F. (2000). Protecting a mobile agent's route against collusions. In *Proceedings of the 6th annual international workshop on selected areas in cryptography* (pp. 215–225). Springer-Verlag.

Anti-Disassembly using Cryptographic Hash Functions

John Aycock, Rennie deGraaf, and Michael Jacobson, Jr.
Department of Computer Science
University of Calgary

About the Authors

John Aycock is an assistant professor at the University of Calgary in the Department of Computer Science. He received a B.Sc. from the University of Calgary, and an M.Sc. and Ph.D. from the University of Victoria. He researches computer security and compilers, and conceived and taught the University's "Computer Viruses and Malware" and "Spam and Spyware" courses.

Rennie deGraaf is a graduate student in Computer Science at the University of Calgary, where he is researching firewalling technology. Other interests include port knocking, malware, honeypots, and the application of cryptography to solve real-world security problems.

Michael Jacobson is an assistant professor at the University of Calgary in the Department of Computer Science. He received a B.Sc. (Hon.) and an M.Sc. from the University of Manitoba and Dr. rer. nat. from the Technical University of Darmstadt. His research interests are cryptography and related applications of computational number theory.

Mailing Address: Department of Computer Science, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4; Phone: +1 403 210 9409; E-mail: {aycock,degraaf,jacobs}@cpsc.ucalgary.ca.

Keywords

Anti-disassembly, code obfuscation, cryptography, cryptographic hash functions, botnets

Anti-Disassembly using Cryptographic Hash Functions

Abstract

Computer viruses sometimes employ coding techniques intended to make analysis difficult for anti-virus researchers; techniques to obscure code to impair static code analysis are called anti-disassembly *techniques. We present a new method of anti-disassembly based on cryptographic hash functions which is portable, hard to analyze, and can be used to target particular computers or users. Furthermore, the obscured code is not available in any analyzable form, even an encrypted form, until it successfully runs. The method's viability has been empirically confirmed. We look at possible countermeasures for the basic anti-disassembly scheme, as well as variants scaled to use massive computational power.*

Introduction

Computer viruses whose code is designed to impede analysis by anti-virus researchers are referred to as *armored* viruses.[1] Armoring can take different forms, depending on the type of analysis being evaded: dynamic analysis as the viral code runs, or static analysis of the viral code. In this paper, we focus on static analysis.

Static analysis involves the tried-and-true method of studying the code's disassembled listing. *Anti-disassembly* techniques are ones that try to prevent disassembled code from being useful. Code using these techniques will be referred to as *disassembly-resistant code* or simply *resistant code*. Although we are only considering anti-disassembly in the context of computer viruses, some of these techniques have been in use as early as the 1980s to combat software piracy (Krakowicz, c.1983).

Ideally, resistant code will not be present in its final form until run time – what can't be seen can't be analyzed. This could involve self-modifying code, which presents problems for static analysis of code (Lo, Levitt, & Olsson, 1995). It could also involve dynamic code generation such as a just-in-time compiler performs (Aycock, 2003).

In this paper, we present a new method of anti-disassembly based on dynamic code generation, which has the following properties:

- It can be targeted, so that the resistant code will only run under specific circumstances. We use the current username as a key for our running example, but any value available to the resistant code (or combinations thereof) with a large domain is suitable, like a machine's domain name. Because this key is derived from the target environment, and is not stored in the virus, our method may be thought of as environmental key generation (Riordan & Schneier, 1998).
- The dynamically generated code is not available in any form, even an encrypted one, where it can be subjected to analysis until the resistant code runs on the targeted machine. Other encryption-based anti-disassembly methods require that the resistant code be available in encrypted form (e.g., Filiol, 2005), in which case it may be subject to analysis.
- Even if the dynamically generated code were somehow known or guessed, the exact key

[1] The techniques we describe can be used by any malicious software (malware), so we use the term “computer virus” in this paper without loss of generality.

used by the resistant code is not revealed.

- It does not rely on architecture-specific trickery and is portable to any platform.

Below, we begin by explaining our anti-disassembly technique and presenting some empirical results. We then look at how the technique might be countered, along with some more entrepreneurial means of deployment.

The Idea

A cryptographic hash function is one that maps each input to a fixed-length output value, such that it is not computationally feasible to reverse the process, nor is it easy to locate two inputs with the same output (Schneier, 1996). Like regular hash functions, a cryptographic hash function is many-to-one.

Our idea for anti-disassembly is to combine a key – here, we use the current username for concreteness – with a "salt" value, and feed the result as input into a cryptographic hash function. The hash function produces a sequence of bytes, from which we extract a subsequence between bytes *lb* and *ub*, and interpret that subsequence as machine code. We will refer to this subsequence as a *run*. The salt value, for this application, is a sequence of bytes chosen by the virus writer to ensure that the desired run appears in the hash function's output when the correct key is used.

A pseudocode example of this idea is shown below: from a high-level point of view, this is what an analyst would be confronted with. The code for a cryptographic hash function is assumed to be available, likely in a system library, and `run` is the code sequence that the virus writer is trying to hide. The task of the analyst is to determine precisely what this code does when executed (the value of `run`) and what the target is (the correct value of `key`).

```
key ← getusername()
hash ← md5(key + salt)
run ← hash_lb…ub
goto run
```

This pseudocode uses the username as a key, and MD5 as the cryptographic hash function (Rivest, 1992); + is the concatenation operator. MD5 is now known to be vulnerable to collisions (Wang, Feng, Lai, & Yu, 2004), but this is irrelevant to our technique: the analyst does not have the MD5 hash, and in any case would be interested in the exact value of the key, not just a key which produces the same hash. In addition, our anti-disassembly technique can be used with any cryptographic hash function, so a different/stronger one can be chosen if necessary.

There are three issues to consider:

1. Having the wrong key. Obviously, if the wrong key value is used, then the run is unlikely to consist of useful code. The resistant code could simply try to run it anyway, and possibly crash; this behavior is not out of the question for viruses. Another approach would catch all of the exceptions that might be raised by a bad run, so that an obvious crash is averted. A more sophisticated scheme could check the run's validity using a checksum (or re-using the cryptographic hash function), but this would give extra information to a code analyst.
2. Choosing the salt. This is the most critical aspect; we suggest a straightforward brute-force search through possible salt values. Normally, conducting a brute-force attack against a cryptographic hash function to find a particular hash value, i.e., a collision, would be out of the question because the hash functions are designed to make this computationally

prohibitive. However, we are only interested in finding a subsequence of bytes in the hash value, so our task is easier. An analysis of the expected computational effort required to find the required salt is presented in the next section.

3. Choosing *lb* and *ub*. These values are derived directly from the hash value, once the desired salt is found.

The salt search is by far the most time-consuming operation, but this need only be done once, prior to the release of the resistant code. The search time can be further reduced in three ways. First, faster machines can be used. Second, the search can be easily distributed across multiple machines, each machine checking a separate part of the search space. Third, the search can be extended to equivalent code sequences, which can either be supplied manually or generated automatically (Joshi, Nelson, & Randall, 2002; Massalin, 1987); since multiple patterns can be searched for in linear time (Aho & Corasick, 1975), this does not add to the overall time complexity of the salt search.

Analysis

In order to find a salt value, we simply compute the cryptographic hash of

```
key + salt
```

for all possible salt values until the hash output contains the required byte sequence (run). In order to speed up the search, we allow the run to begin in any position in the hash output.

Approximately half of the output bits of a cryptographic hash function change with each bit changed in the input (Schneier, 1996); effectively, we may consider the hash function's output to change randomly as the salt is changed. Given that, the probability of finding a particular *b*-bit run in a fixed position of an *n*-bit output is the ratio of the bits not in the run to the total number of bits: $2^{n-b}/2^n$, or $1/2^b$. The expected number of attempts would then be 2^{b-1}. Furthermore, because only the salt is being changed in the brute-force search, this implies that we would need *b-1* bits of salt for a *b*-bit run.

If we allow *lb*, the starting position of the run, to vary, the expected number of attempts will be reduced by a factor equal to the number of possible values of *lb*. If we index the starting position at the byte level, then there are $m = (n-b)/8$ possible starting positions. The probability of finding the *b*-bit run increases to $m/2^b$, and the expected number of attempts becomes $2^{b-1}/m$. Similarly, if we index at the bit level, there are *n-b* starting positions and the expected number of attempts reduces further to $2^{b-1}/(n-b)$.

Notice that the computational effort depends primarily on the length of the run, not the length of the hash function output. The length of the hash function only comes into play in reducing the expected number of attempts because the number of possible values for *lb*, the starting point of the run, depends on it.

We only discuss the case of single runs here, but this technique trivially extends to multiple runs, each with their own salt value. Because the salt computation for each run is independent of the others, the total effort required for multiple-run salt computation scales linearly. If the computational effort to compute the salt for one run is *X*, then the effort for one hundred runs is *100X*.

As an example of salt computation, suppose we want our run to consist of a single Intel x86 relative jump instruction. This instruction can be encoded in five bytes, so we need to find a salt that, when concatenated to the key, yields a hash value containing this five-byte run starting in any position.

The MD5 hash function has 128-bit outputs, so if we index the run at the byte level, there are 11 possible values for *lb*. The expected number of attempts to find the run is therefore

$$2^{39} / 11 < 2^{36}.$$

If we instead index at the bit level, there are 88 possible values for *lb* and the expected number of attempts is

$$2^{39} / 88 < 2^{33}.$$

Using a 160-bit hash function such as SHA-1 yields $2^{39} / 15$ and $2^{39} / 120$ when indexing *lb* at the byte and bit levels, respectively. In all cases, the computation can be done in only a few hours on a single modern desktop computer.

It is feasible to use this method to find runs slightly longer than five bytes, but the computational effort adds up very quickly. For example, to find an eight-byte run using SHA-1 and indexing *lb* at the bit level would require roughly $2^{63} / 120 > 2^{56}$ attempts. A special-purpose, massively parallel machine would likely be required to find the run in this case, as the computational effort involved is roughly equivalent to that required to break the DES block cipher, for which such hardware was also required (Electronic Frontier Foundation, 1998).

Empirical Results

To demonstrate the feasibility of this anti-disassembly technique, we searched for the run (in base 16)

```
e9 74 56 34 12
```

These five bytes correspond on the Intel x86 to a relative jump to the address 12345678_{16}, assuming the jump instruction starts at address zero.

Algorithm	*Key*	*Salt*	*Search Time (s)*
MD5 (128 bits)	aycock	55b7d9ea16	39615
	degraaf	a1ddfc1910	47650
	foo	e6500e0214	60185
	jacobs	9ac1848109	28723
	ucalgary.ca	4d21abe205	18220
	Average		44746
SHA-1 (160 bits)	aycock	07e9717a09	36584
	degraaf	0d928a260e	55424
	foo	2bc680de1e	120472
	jacobs	ca638d5e06	24958
	ucalgary.ca	585cc614	325
	Average		47552

Table 1: Brute-force salt search for a specific five-byte run

The search was run on an AMD AthlonXP 2600+ with 1 GB RAM, running Linux 2.6. We tested five different keys with one- to five-byte salts, sequentially searching through the possible salt

values.[2] Table 1 shows the results for two cryptographic hash functions, MD5 and SHA-1. For example, the salt "07e9717a09," when concatenated onto the key "aycock," yields the SHA-1 hash value

```
ef 6d f4 ed 3b a1 ba 66 27 fe e9 74 56 34 12 a2 d0 4f 48 91
```

Numbering the hash value's bytes starting at zero, our target run is present with *lb = 10* and *ub = 14*.

Another question is whether or not every possible run can be produced. Using the key "aycock," we were able to produce all possible three-byte runs with three bytes of salt, but could only produce 6% of four-byte runs with a three-byte salt. With a four-byte salt, we were able to generate four-byte runs which covered between 99.999-100% of the possible combinations – this was checked with five different keys and three different cryptographic hash functions. (Our test system did not have sufficient memory to record coverage data for five-byte runs in a reasonable amount of time.) The four-byte run data are shown in Table 2.

Algorithm	*Key*	*Runs Found*	*Runs Not Found*
MD5 (128 bits)	aycock	4294936915	30381
	degraaf	4294937044	30252
	foo	4294936921	30375
	jacobs	4294937188	30108
	ucalgary.ca	4294936946	30350
	Average	4294937003	30293
SHA-1 (160 bits)	aycock	4294966707	589
	degraaf	4294966733	563
	foo	4294966660	636
	jacobs	4294966726	570
	ucalgary.ca	4294966769	527
	Average	4294966719	577
SHA-256 (256 bits)	aycock	4294967296	0
	degraaf	4294967296	0
	foo	4294967296	0
	jacobs	4294967296	0
	ucalgary.ca	4294967296	0
	Average	4294967296	0

Table 2: Generation of possible four-byte runs using a four-byte salt

These results tend to confirm our probability estimate from the last section: *b*-bit runs need *b-1* bits of salt. Four-byte runs are of particular interest for portability reasons, because RISC instruction sets typically use instructions that are four bytes long; this means that at least one RISC instruction can be generated using our technique. One instruction may not seem significant, but it is sufficient

[2] For implementation reasons, we iterated over salt values with their bytes reversed, and didn't permit zero bytes in the salts.

to perform a jump anywhere in the address space, perform an arithmetic or logical operation, or load a constant value – potentially critical information that could be denied to an analyst.

Countermeasures

An analyst who finds some resistant code has several pieces of information immediately available. The salt, the values of *lb* and *ub*, and the key's domain (although not its value) are not hidden. The exact cryptographic hash function used can be assumed to be known to the analyst, too – in fact, resistant code could easily use cryptographic hash functions already present on most machines.

There are two pieces of information denied to an analyst:

1. The key's value. Unless the key has been chosen from a small domain of values, then this information may not be deducible. The result is that an analyst may know that a computer virus using this anti-disassembly technique targets someone or something, but would not be able to uncover specifics.
2. The run. If the run is simply being used to obscure the control flow of the resistant code, then an analyst may be able to hazard an educated guess about the run's content. Other cases would be much more difficult to guess: the run may initialize a decryption key to decrypt a larger block of code; the entire run may be a "red herring" and only contain various NOP instructions.

Note that even if the run is somehow known to an analyst, the cryptographic hash function cannot be reversed to get the original key; this is a property of the cryptographic hash function (Schneier, 1996). At best, the analyst could perform their own brute-force search to determine a set of possible keys (recall that the hash function is many-to-one). However, the analyst also knows the salt and the domain of the key, so given the run, the analyst can find the key by exhaustively testing every possible value. This underscores the point that the key domain must be sufficiently large to preclude such a brute-force analysis – our example in the last section of using usernames as keys would likely not prevent this.

Whether or not every last detail of the resistant code can be found out is a separate issue from whether or not a computer virus using resistant code can be detected. In fact, there is malware already that can automatically update itself via the Internet, like Hybris (F-Secure, 2001), so complete analysis of all malware is already impossible.

Fortunately for anti-virus software, computer viruses using the technique we describe would present a relatively large profile which could be detected with traditional defenses, including signature-based methods and heuristics (Szor, 2005). Precise detection does not require full understanding.

Enter the Botnet

What if the computing power available for a brute-force salt search were increased by five orders of magnitude over the computer we used for our experiments? Few organizations have that much computing power at their fingertips, but a few individuals do. A *botnet* is a network of malware-controlled, "zombie" machines that executes commands issued via Internet Relay Chat (IRC) channels (Cooke, Jahanian, & McPherson, 2005). These have been used for sending spam and distributed denial-of-service attacks (Cooke et al., 2005), but they may also be viewed as very large-scale distributed computing frameworks which can be used for malicious purposes.

If a virus writer wants to armor a virus using the anti-disassembly technique described here, especially for long runs with many instructions, a botnet may be used for salt computation. A naïve

salt computation on a botnet would involve partitioning the salt search space between machines, and the key and desired run would be available to each machine. Using the earlier Intel x86 relative jump example, for instance, four zombie machines in a botnet could each be given the desired key (e.g., ``aycock'') and run (e974563412) and a four-byte salt search could be divided like so:

Zombie 1 00000000…3fffffff

Zombie 2 40000000…7fffffff

Zombie 3 80000000…bfffffff

Zombie 4 c0000000…ffffffff

Having the virus writer's desired key and run on each zombie machine would not be a bad thing from an analyst's point of view, because locating any machine in the botnet would reveal all the information needed for analysis.

A more sophisticated botnet search would do three things:

1. Obscure the key. A new key, *key´*, could be used, where *key´* is the cryptographic hash of the original key. The deployed resistant code would obviously need to use *key´* too.
2. Supply disinformation. The virus writer may choose *lb* and *ub* to be larger than necessary, to mislead an analyst. Unneeded bytes in the run could be NOP instructions, or random bytes if the code is unreachable.
3. Hide the discovery of the desired run. Instead of looking for the exact run, the botnet could simply be used to narrow the search space. A weak checksum could be computed for *all* sequences of the desired length in the hash function's output, and the associated salts forwarded to the virus writer for verification if some criterion is met. For example, the discovery of our five-byte run in the "Empirical Results" section could be obliquely noted by watching for five-byte sequences whose sum is 505.

This leaves open two countermeasures to an analyst. First, record the *key´* value in an observed botnet in case the salt is collected later, after the virus writer computes and deploys it – this would reveal the run, but not the original key. Second, the analyst could subvert the botnet, and flood the virus writer with false matches to verify. The latter countermeasure could itself be countered quickly by the virus writer, however, by verifying the weak checksum or filtering out duplicate submissions; in any case, verification is a cheap operation for the virus writer.

Related Work and Conclusion

There are few examples of strong cryptographic methods being used for computer viruses – this is probably a good thing. Young and Yung (1996) discuss cryptoviruses, which use strong cryptography in a virus' payload for extortion purposes. Riordan and Schneier (1998) mention the possibility of targeting computer viruses, as does Filiol (2005).

Filiol's work is most related to ours: it uses environmental key generation to decrypt viral code which is strongly-encrypted. Neither his technique nor ours stores a decryption key in the virus, instead finding the key on the infected machine. A virus like the one Filiol proposes hides its code with strong encryption, carrying the encrypted code around with the virus. In our case, however, the code run never exists in an encrypted form; it is simply an interpretation of a cryptographic hash function's output. Our technique is different in the sense that the ciphertext is not available for

analysis.

The dearth of strong cryptography in computer viruses is unlikely to last forever, and preparing for such threats is a prudent precaution. In this particular case of anti-disassembly, traditional defenses will still hold in terms of detection, but full analysis of computer viruses may be a luxury of the past. For more sophisticated virus writers employing botnets to find salt values and longer runs, proactive intelligence gathering is the recommended defense strategy.

Acknowledgments

The first and third authors' research is supported in part by grants from the Natural Sciences and Engineering Research Council of Canada. Karel Bergmann made helpful comments on an early version of this paper, as did the anonymous referees.

References

Aho, A. V., and Corasick, M. J. (1975). Efficient string matching: an aid to bibliographic search. Communications of the ACM, 18(6), 333–340.

Aycock, J. (2003). A brief history of just-in-time. ACM Computing Surveys, 35(2), 97–113.

Cooke, E., Jahanian, F., & McPherson, D. (2005). The zombie roundup: Understanding, detecting, and disrupting botnets. In USENIX SRUTI Workshop.

Electronic Frontier Foundation (1998). Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design. O'Reilly.

F-Secure (2001). F-Secure virus descriptions: Hybris. Retrieved 5 January, 2006, from http://www.f-secure.com/v-descs/hybris.shtml.

Filiol, E. (2005). Strong cryptography armoured computer viruses forbidding code analysis: The Bradley virus. In Proceedings of the 14th Annual EICAR Conference, pages 216–227.

Joshi, R., Nelson, G., & Randall, K. (2002). Denali: a goal-directed superoptimizer. In Proceedings of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation, pages 304–314.

Krakowicz (c. 1983). Krakowicz's kracking korner: The basics of kracking II. Retrieved 5 January, 2006, from http://www.skepticfiles.org/cowtext/100/krckwczt.htm.

Lo, R. W., Levitt, K. N., & Olsson, R. A. (1995). MCF: a malicious code filter. Computers & Security, 14, 541–566.

Massalin, H. (1987). Superoptimizer: a look at the smallest program. In Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems, pages 122–126.

Riordan, J., & Schneier, B. (1998). Environmental key generation towards clueless agents. In Mobile Agents and Security (LNCS 1419), pages 15–24.

Rivest, R. (1992). The MD5 message-digest algorithm. RFC 1321.

Schneier, B. (1996). Applied Cryptography. Wiley, 2nd edition.

Szor, P. (2005). The Art of Computer Virus Research and Defense. Addison-Wesley.

Wang, X., Feng, D., Lai, X., & Yu, H. (2004). Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199. Retrieved 5 January, 2006, from http://eprint.iacr.org/.

Young, A., & Yung, M. (1996). Cryptovirology: Extortion-based security threats and countermeasures. In IEEE Symposium on Security and Privacy, pages 129–141.

Computer Crime Revisited: The Evolution of Definition and Classification

Sarah Gordon, Symantec & Richard Ford, Florida Tech.

About Author(s)

Sarah Gordon is a Research Scientist at Symantec Security Response. Contact Details: Symantec Security Response, 2500 Broadway, Santa Monica, CA 90494 USA, e-mail firstinitiallastname_at_symantec.com

Richard Ford is an Associate Professor at the Florida Institute of Technology
Contact Details: Dept. of Computer Sciences, Florida Tech., 150 W. University Blvd, Melbourne, FL 32901, USA, phone +1-321-6747473, e-mail firstinititallastname_at_fit.edu

Keywords

Cybercrime, Crimeware, Computer Crime, Cyberstalking, Fraud

Computer Crime Revisited: The Evolution of Definition and Classification

Abstract

The idea of Cybercrime is not new, yet there is significant confusion amongst academics, computer security experts and users as to the extent of real Cybercrime. In this paper, we explore the breadth of computer-based crime, providing a definition of the emerging terms "Cybercrime" and "crimeware". We then divide Cybercrime into two distinct categories: Type I Cybercrime, which is mostly technological in nature, and Type II Cybercrime, which has a more pronounced human element. We then use two case studies to illustrate the role of crimeware in different types of Cybercrime, and offer some observations on the role of cognition in the process of Cybercrime. Finally we provide several suggestions for future work in the area of Cybercrime.

Introduction

Discussions of Cybercrime can be found in diverse sources including academic journals, generalist computer magazines, newspaper articles, and online; it has been the subject of movies, television programs and radio broadcasts. However, despite an apparent acceptance of and familiarity with the term, there exist dramatically varied views of what Cybercrime *is*. This lack of definitional clarity is problematic as it impacts every facet of prevention and remediation. In addition, research shows that the number of people and businesses impacted by various types of perceived cybercrime is growing with no signs of declining (ISTR, 2005; Chawki, 2005; ZD, 2005).

In this paper we examine various dimensions of Cybercrime; after examining some of these definitions, we offer a more inclusive definition before delineating these crimes into two subtypes. Further, we define the term "crimeware", now in common usage but with varied and often context-based definition, and illustrate this definition's breadth of similarities and differences with existing usages. Finally two case studies are provided, illustrating the varying effects of education and perception with respect to Cybercrime on home users.

Definitions

Despite the fact that the word "Cybercrime" has entered into common usage, many people would find it hard to define the term precisely. Furthermore, there is no catch-all term for the tools and software which are used in the commission of certain online crimes. In the next two sections, we will attempt to rigorously define Cybercrime, and formalize an emerging term, crimeware, which is an inclusive term for the many different Trojans, Viruses, Bots, Spyware and Worms which are instrumental in facilitating certain Cybercrimes.

Cybercrime

Like traditional crime, Cybercrime has many different facets and occurs in a wide variety of scenarios and environments. Current definitions of Cybercrime have evolved experientially. They differ depending upon the perception of both observer/protector and victim, and are partly a function of computer-related crimes geographic evolution. For example, The Council of Europe's Cybercrime Treaty uses the term "Cybercrime" to refer to offences ranging from criminal activity against data to content and copyright infringement (Krone, 2005). However, (Zeviar-Geese, 1997-98) suggest that the definition is broader, including activities such as fraud, unauthorized access, child pornography, and cyberstalking. The United Nations Manual on the Prevention and Control of Computer Related Crime includes fraud, forgery, and unauthorized access (United Nations, 1995) in its cybercrime definition.

As you can see from these three definitions, Cybercrime can occur across a broad spectrum. In many ways, our argument regarding Cybercrime is similar to our previous argument concerning the utility of the word "cyberterrorism" (Gordon & Ford, 2004). In the case of cyberterrorism it is our belief that the term itself is misleading in that it tends to create a vertical representation of a problem that is inherently horizontal in nature. Similarly, a criminal will not care whether a crime is "cyber" in nature or not; instead, all methods available will be exploited.

Given this position, we believe there are significant benefits to deleting the word from the lexicon entirely! However, given that this is not likely to occur, the next best thing is to attempt to define the word as meaningfully as possible. Unfortunately, modelling cybercrime definition upon existing categories in work such as [Parker, 1988] is problematic as existing work tends to be descriptive rather than based upon a theoretical framework. With this in mind, we define Cybercrime as:

> "any crime that is facilitated or committed using a computer, network, or hardware device".

The computer or device may be the agent of the crime, the facilitator of the crime, or the target of the crime; indeed, the crime can take place on the computer alone, or in other non-virtual locations. Given the breadth of this definition it is beneficial to subdivide cybercrime into two distinct types; thus we define operationally for the purpose of this research Type I and Type II Cybercrime. An additional advantage of this approach is that it is easy for researchers to define the topic too narrowly. By explicitly highlighting these two facets of Cybercrime, we hope to provide additional emphasis on the breadth of the issue.

Further, our goal is not to legally define Cybercrime – such a definition is beyond the scope of this paper. Instead, we attempt to create a conceptual framework which lawmakers can use in order to create legal definitions which are meaningful from a technical and societal perspective. We recognize that current legal definitions of Cybercrime vary drastically between jurisdictions; however, if technicians in the field worldwide can adequately grasp the nuances of electronic crime, more cohesive legal definitions may result.

Under our proposed scheme, Type I cybercrime has the following characteristics:

1. It is generally a singular, or discrete, event from the perspective of the victim.
2. It often is facilitated by the introduction of crimeware programs such as keystroke loggers, viruses, rootkits or Trojan horses into the user's computer system
3. The introductions can, but may not necessarily be, facilitated by vulnerabilities.

A single event or discrete instance, from the user's perspective, might look something like this:

1. The user goes online to perform a task, i.e. access the WWW, or read/reply to e-mail.
2. User takes action which then allows the criminal access to information (entering personal information on the look-a-like site, (or) clicks on some object resulting in the download of a Trojan or keystroke logger.
3. This information is used by the attacker.
4. The user becomes aware of the crime – this is the single event from the perspective of the user. This usually occurs much later in the lifecycle of the Cybercrime.
5. The crime is investigated and resolved.

This type of Cybercrime requires that data be protected from traditional threats such as viruses and worms, but also that users be cognizant of the concept of "vulnerabilities". This is a huge "thought change" required in the user population.

Vulnerabilities are often found in COTS software; for example, in 2005, Microsoft documented several key vulnerabilities in its popular Internet Explorer application. Criminals controlling a web site may use computer code capable of exploiting such a vulnerability in a Web Browser to place a back-door program on the computer of a visiting user.

Examples of this type of cybercrime include but are not limited to phishing attempts, theft or manipulation of data or services via hacking or viruses, identity theft, and bank or e-commerce fraud based upon stolen credentials.

Type II cybercrime, at the other end of the spectrum, includes, but is not limited to activities such as cyberstalking and harassment, child predation, extortion, blackmail, stock market manipulation, complex corporate espionage, and planning or carrying out terrorist activities online. The characteristics of Type II cybercrime are that:

1. It is generally facilitated by programs that do not fit under the classification crimeware. For example, conversations may take place using IM (Instant Messaging) clients or files may be transferred using the FTP protocol.
2. There are generally repeated contacts or events from the perspective of the user.

A series of events in the lifecycle of a Type II Cybercrime might look something like this:

1. User(a) goes online to see what she can find out about llama farming.
2. User(a) decides to participate in on-line forum about llama farming.
3. User(b) sees User(a), watches her participation in the forum for several days, responds to some of her comments.
4. User(b) then sends a request for private chats using a common Instant Messaging client.
5. User(a), being familiar with User(b) via the on-line forum, responds positively and they begin to chat daily as well as participate in the Forum. This is a period known as instilling trust.
6. After several interactions User(a) reveals that she is single, likes llamas, has a quarter of a million dollars available to start a llama farm, and that she likes to go to concerts. She tells him her real name is Jenny.
7. User(b) asks User(a) to meet in person, and go to a concert.
8. User(a) becomes suspicious when User(b) will not give his contact information other than on-line information, and she refuses.
9. User(b) becomes irrational and begins to post false claims against User(a) in the on-line forum, accusing her of fraud, and of being there to pick up men, not to find others interested in llamas. He posts her home number. He also goes onto other forums posing as User(a), and leaves messages asking for dates – leaving her real phone number and real name.
10. User(a) tries to defend herself in the Forum, and asks User(b) privately to stop, using IM. She begins to get numerous e-mails about the dates she requested – and realises then that someone is impersonating her online. She confronts User(b) with her suspicions.

11. User(b) becomes more irrational and begins to make hang-up and harassing phone calls to User(a). User(a) becomes afraid for her safety.
12. The telephone company and the local police become involved.
13. User(a) files charges against User(b), who is, it is later learned, a former child-pornographer with links to organized crime, under investigation for the disappearance of three women he allegedly met on the Internet.

While such an exchange may seem far-fetched, cyberstalking is a very real problem in today's online community (Bocji, 2005). As such, it clearly is related to – but different from – Type I Cybercrimes which are more technical in nature.

Part of the reasoning behind this classification is that Cybercrime really presents a continuum ranging from crime which is almost entirely technological in nature and crime which is really, at its core, entirely people-related. Consider, for example, a fraud carried out via email where the user is directly and simply asked to send money to a particular physical address in return for some service which never materializes. At its core, this fraud would work via regular paper mail or telephone. As such, it is really not a technological issue, though the perpetrator can (and frequently does) use certain attributes of technology to his or her advantage. At the other end of the scale, the user whose machine is penetrated but suffers no financial loss has not really participated in the Cybercrime – the crime is purely technological in nature.

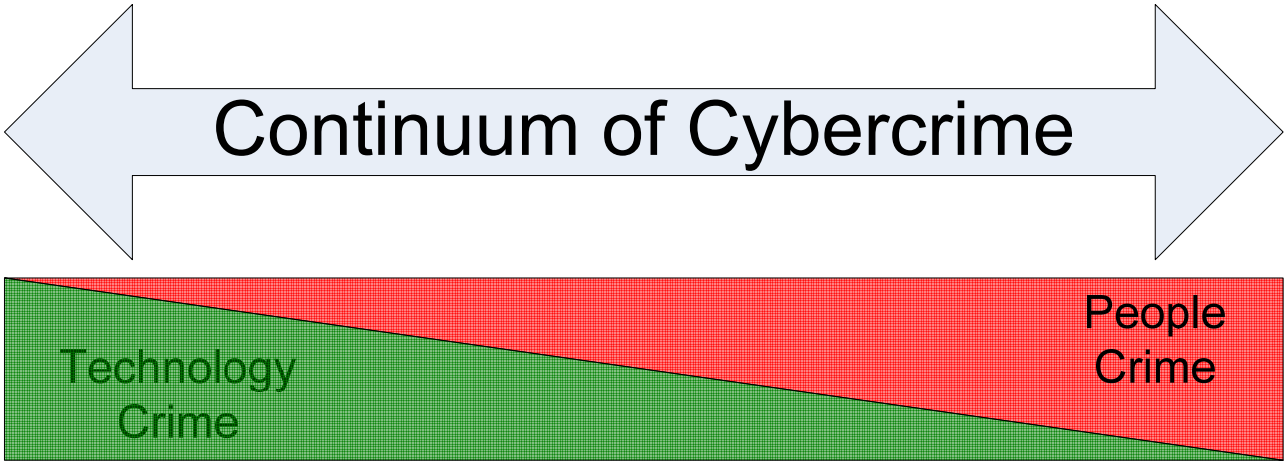


Figure 1: The Continuum of Cybercime. Areas defined as Cybercrime are very broad in nature – some crimes have only a peripheral cyber element, whereas others exist only in the virtual world.

Similarly, there are likely to be very few events which are *purely* Type I or Type II; these types represent either end of a continuum. The important fact is to recognize the breadth of the scale; it is all too easy to ignore the end of the scale which one is least accustomed to. For example, a traditional investigator may be more capable of investigating crimes that are more people-centric than technological; similarly, computer security experts are more likely to focus their efforts on issues they see as technological, when simple technological countermeasures could provide significant protection from more people-centric crimes.

Crimeware

The software used in Cybercrime is sometimes referred to as crimeware (see, for example, (Wikipedia, 2006) for a common usage definition). We define crimeware as software that is:

1. used (directly or indirectly) in the commission of the criminal act.

2. not generally regarded as a desirable software application from the perspective of the computer user.
3. not involuntarily enabling the crime.

The reasoning behind this definition is to exclude legitimate programs which may be leveraged by an attacker. For example, a browser which has a vulnerability in it is not Crimeware. In addition, note that *the type of crime carried out is not specified.* This is an important distinction from other definitions of crimeware, which frequently limit its scope to just software used for financial crime. This is an artificial distinction, as it defines a program based upon how it is used not on its content. These issues are further discussed below.

Like Cybercrime itself, the term crimeware covers a broad spectrum. However, it is important to remember that not all software used in the commission of a computer-based or computer-facilitated crime can be legitimately termed crimeware. For example, while an IM client may be used in the commission of a Cybercrime, the IM application software itself is not considered crimeware. FTP clients may be used in the commission of crimes; however, they are not considered crimeware. Crimeware does, however, include programs which may be classified as bots, keystroke loggers, spyware, backdoors and Trojan horses. Additionally, some cybercrime may involve both crimeware and legitimate programs.

Example	Type	Software	Crimeware
Phishing	I	Mail client	No
Identity Theft	I	Keylogger, Trojan	Yes
Cyberstalking	II	Email Client, Messenger Clients	No
DDoS	I	Bots	Yes
Cyberterrorism (communication)	II	Steganography, Encryption, Chat Software	No

Table 1: Cybercrimes by Type. Examples of Different Cybercrimes by type, examining the software used in each case.

Case Study: Type I Cybercrime

Kobe[1], a middle school instructor, recently fell victim to a phishing attack. He was using e-Bay to sell one of his vehicles, and he found a suitable buyer within several days. The buyer paid for the vehicle, Kobe received payment, and removed the listing from e-Bay.

He was somewhat puzzled when he logged into his e-Bay account and was informed he had "one item for sale". He looked at the page, and sure enough, there was the vehicle – the same one he had

[1] Names and places in this section have been changed in order to respect the privacy of the individuals involved.

just sold – for sale. Then he noticed something else wrong – very wrong. The e-mail address that was listed for his contact information was not his. It was very similar, so much so that most people would not notice the change. Kobe e-mailed the "seller", and offered to buy the vehicle, just to see what the response would be. The "seller" thought Kobe was just another buyer, and when Kobe offered to buy the vehicle, the seller made arrangements for the money to be sent. As it turned out, the "seller" was located in Chicago. Kobe gave the FBI the information, and they tracked down the fraudsters.

How did the fraudsters gain access to Kobe's account in the first place? A phishing e-mail stating his account had been compromised asked him to click on a URL to go to his e-Bay account and validate his ID. He clicked on the URL and was taken to a page that looked identical to his e-Bay login page, and he was asked to type in his account information. The criminals used that information to log into his legitimate account and change the contact phone number.

The software used to commit and enable this crime consists of both non-crimeware and crimeware. The non-crimeware programs are those which are used daily by many people in the course of doing business on the Internet: e-mail and a browser. The crimeware program used in this case was executable code that took Kobe to the look-a-like WWW site. The result of the use of the crimeware was the obtaining of Kobe's confidential information, and the resultant placement of a "copycat" for sale ad, designed to lure users into repeatedly purchasing non-existent goods. While the criminals attacked many people using the look-a-like site phishing scam, from the perspective of each user, this crime was a "one time event"[2]. And, while from the perspective of e-Bay, the cybercrime represented multiple frauds, from the perspective of the other victims – those who sent money to the fraudsters, this was also a one-time event.

Case Study: Type II Cybercrime

In some cases, determining whether or not actions are "cybercrime" requires determining if the actions are objectionable. This can depend on the cognitions of the potential victim, or may be clearly recognized according to law or statute.

Consider the case of Roger and Hannah. Several months after the end of their relationship Roger received an e-mail from Hannah. It seemed to be harmless, stating simply that she had been thinking about him and missed him. Roger did not respond to the e-mail, as he did not wish to appear interested in resuming the relationship. Over the next several months, Hannah continued to e-mail Roger from time to time, saying that she knew she should leave him alone, but couldn't. Initially, Roger was not too concerned about Hannah's behaviour; however, when the e-mails persisted the entire year, he became concerned and sent Hannah a strongly worded e-mail telling her he did not want any further contact with her, and after an initial refusal to comply, Hannah did stop all contact.

Discussion

The definition of Cybercrime and the differentiation of types of Cybercrime are extremely important for several reasons. Definitions provide researchers with a common language, necessary for sound collaboration (or even meaningful discussion). Furthermore, definitions help determine the scope of the problem to be addressed, and are necessary for clear communication about a

[2] A user may receive many "individual event" e-mails – but each is considered a discrete event.

subject. The lack of clear definitions is exemplar of immature and unscientific approach to the problem. This is not surprising, given the relative immaturity of the security and in particular antivirus industries (Kuhn, 1992; Gordon, 2005a).

Unfortunately, the definitions of crimeware currently in use by both generalist and specialist populations adds to the confusion by failing to delineate correctly between legitimate software which is used to facilitate a crime and by software whose primary purpose is to execute a crime. It is worth noting at this juncture that software does not have "intent"; instead, we can only attempt to make a best guess as to the intent of the programmer of the software (Gordon, 2005b)

Consider these two definitions:

> **(a1)** Software designed to steal personal information or perform some other illegal operation. It is malicious software that causes a crime to be committed. See warez and malware.
>
> **(a2)** Software that helps someone perform an unwanted or illegal act via the computer. Programs and documentation that enable less technical people to set up their own spam, virus or phishing attacks are crimeware, essentially a software development kit for scoundrels. The good news is that the documentation is probably as horrid as that of most popular commercial software. (PC Magazine, 2005)

And:

> **[b]** Crimeware is a relatively new term that is used to describe software used to commit crime. (Informit, 2005)

We have already shown that software that helps people to perform an illegal act via the computer is not necessarily bad. The claim that software defined as crimeware *causes* a crime to be committed is unsupported; clearly, it is the cybercriminal makes the decision to commit the crime. However, it is the opening sentence of a2 that is more cause for concern, stating "software the helps someone perform" a crime. Definition [b] is problematic in that there are many types of software used to commit crime, as demonstrated by our Case Studies, and considering all of these programs "crimeware" would be inappropriate and counterproductive – essentially classifying almost any program as crimeware under certain circumstances. Thus, these types of definitions are of limited use as they tend to unrealistically broaden the range of programs that could operationally be considered crimeware, robbing the term of meaning.

Another type of definition, from (Davis, Wright & Tremaine, 2005) refers to another type of software to help define Crimeware, stating "Crimeware is similar to spyware in that it monitors a user's online behaviour; however, crimeware programs have been modified for the purpose of stealing a user's personal information". This type of overgeneralization tends to limit the scope of the problem in that it considers only monitoring programs to be crimeware, and perhaps more problematically, it does not differentiate between programs that monitor legitimately versus those that monitor illegitimately.

At the very opposite end of this spectrum, (StudyCrime, 2005) limits crimeware to those programs used to commit financial crime, as opposed to other types of malware which may not aid in financial crime. "Crimeware is a term coined by Peter Cassidy, Secretary General of the Anti-Phishing Working Group, to distinguish computer programs (and coordinated, interlocking sets of programs) that are designed specifically to animate financial crime from other kinds of malevolent code packages." This narrow definition limits the scope of the problem artificially, it does not even

address the issue of programs which are shown to be behaviourally malware (such as a program designed to exploit a vulnerability and obtain root access remotely) and which *could* be used to commit financial or other cybercrime; rather, it excludes them!

In light of the issues introduced by the current span of operational definitions of "crimeware", we consider first the Case Study of Roger and Hannah detailed in the previous section. Zona, Pallarea & Lane (1998) and Zona, Sharma & Lane (1993) describe a typology of cyberstalking based on the relationship between the victim and offender that is consistent with the case of Roger and Hannah. Classified as "Simple Obsessional", such cases "*typically involve a victim and a perpetrator who have a prior relationship. This group comprises the largest of the categories (47 percent), and also poses most threat to the victim. The motivation behind this may be coercion to re-enter a relationship, or revenge aimed at making the life of the former intimate uncomfortable through the inducement of fear."*

Additionally, this type of behaviour has the potential to develop into something far more serious. According to the (DOJ, 1999) "while some conduct involving annoying or menacing behaviour might fall short of illegal stalking, such behaviour may be a prelude to stalking and violence and should be treated seriously." (Helpguide, 2005) concurs: "Stalking is unpredictable and should always be considered dangerous."

Indeed, the question of whether or not unwanted (but otherwise benign) e-mail constitutes stalking is addressed clearly by ((DOJ, 2005); see also the following statues: Michigan, (MSA § 28.643(8)E,vi) , Oklahoma, (21 Okl. St.§ 1173, F 4f), Alaska (AK St. 11.41.270)), all of which place "unwanted e-mail contact" into the category of cyberstalking. Roger did not consider Hannah's actions "cyberstalking"; however, the question remains, at what point does "unwanted contact" constitute "cyberstalking" or harassment? In any case, the resolution of situations involving unwanted contact is sometimes not benign; as evidenced by the following examples:

A San Diego college student's actions reported in (DOJ, 1999): "*An honours student from the University of San Diego terrorized five female university students over the Internet for more than a year. The victims received hundreds of violent and threatening e-mails, sometimes receiving four or five messages a day. The graduate student, who has entered a guilty plea and faces up to six years in prison, told police he committed the crimes because he thought the women were laughing at him and causing others to ridicule him. In fact, the victims had never met him."*

While Cybercrime is only now gaining high visibility, these cases are not exceptions. There have been cases of the Internet being used to facilitate crime throughout the past decade. For example, in 1999, a 50 year old man who had used the Internet to solicit the rape of a woman who had rejected his romantic advance pled guilty to one count of stalking and six counts of soliciting sexual assault. His actions included impersonating his 28 year old victim in various Internet chat rooms and online bulletin boards, posting messages allegedly from her stating that she fantasized of being raped, and providing her address and telephone number. On at least six occasions, sometimes in the middle of the night, men knocked on the woman's door saying they wanted to rape her (DOJ, 1999).

In all of these cases, the software used to commit or enable the Cybercrimes should not be classified as crimeware. In the case of Roger and Hannah, e-mail and instant messaging software was used to do what are normal everyday actions; the sending of e-mail. The student from San Diego crossed the line even further in the commission of his Cybercrime; however, he still did not use crimeware. Finally, the man eventually convicted for stalking and sexual assault used e-mail programs, bulletin board software, and chat clients to post messages, send e-mail and chat – something most of us do every day using the same types of legitimate software he used to commit crime.

Clearly the skill-set needed to investigate Type I Cybercrime differs greatly from the skill-set needed to investigate Type II Cybercrime. Additionally, while there *is* some defense from Type II Cybercrime afforded by technology, the primary defense is currently more human-centric.

Topics for Future Research

This work explores the naturally occurring division between the varied types of Cybercrime. Future research could build on this work by exploring other types of crime that fits, or that could fit, into these divisions. This might give us some insights into what the future of Cybercrime might entail, and by examining both the technical and "people" aspects, it would help us to approach the problem from a more holistic perspective. Compiling a list of program types that would fit under the definition of crimeware would be useful as well. It would be interesting to see how many Cybercrimes actually include crimeware. It is our suspicion that that there are fewer people committing Type I Cybercrimes than is often believed, but that the nature of the crime produces an artificially high estimation of the scope of the problem in terms of perpetrators. It would be interesting to know if similar division occurs in environments that are fully virtual; for example, Second Life.

Conclusion

In this paper, we have examined some of the existing definitions of Cybercrime and crimeware, and found there to be significant lack of clarity in their common usage. To address this, we have proposed a more concise definition of these terms, and have further subdivided the area of Cybercrime into two separate areas to facilitate an understanding of the crimes technological and human dynamic. This understanding is of critical importance, as organizations tasked with defending populations against Cybercrime must begin to at least consider all the crimes within this continuum, and designate appropriate resources to prevent, defend against, and investigate Cybercrime. This is especially important as new laws which address "Cybercrime" begin to take effect at a Federal and State level.

Our definition and separation of cybercrime according to its primary "human" or "not human" factors can be a first step in future research that begins to map these crimes according to a variety of factors. For example, development of a matrix similar to that proposed in earlier Cyberterrorism work could result in an even greater understanding of Cybercrime, possibly enabling researchers to accurately and precisely predict the direction of future Cybercrime. It is to be hoped that by studying Cybercrime with a more holistic perspective, novel solutions to both types of Cybercrime can be created.

Finally, we believe that the situation with Cybercrime and crimeware is rapidly evolving. By attempting to view the problem more inclusively, it should be possible to foresee new develops and take steps toward remediation rapidly. Narrowing or ignoring the problems will create the perfect environment for the Cybercriminal to flourish, undermining the perceived stability and reliability of electronic systems worldwide.

References

Bocij, P. (2005) Reactive stalking: a new perspective on victimisation. The British Journal of Forensic Practice, Feb 2005

Chawki, M. (2005). Cybercrime in France: An Overview. Computer Crime Research Center. December, 2005. Downloaded January 23rd 2006 from: http://www.crime-research.org/articles/cybercrime-in-france-overview/.

Davis E., Wright H., & Tremaine C. (2005). Beyond Phishing: Pharming and Crimeware Attacks. Downloaded Jan 23rd 2006 from: http://www.privsecblog.com/archives/phishingpharming-53-beyond-phishing-pharming-and-crimeware-attacks-.html

DOJ, (1999). Cyberstalking: A New Challenge for Law Enforcement and Industry" A Report from the Attorney General to the Vice President (August 1999) Downloaded January 23rd 2006 from http://www.usdoj.gov/criminal/cybercrime/cyberstalking.htm.

DOJ, (2005). H.R. 3402 Department of Justice Appropriations Authorization Act, Fiscal Years 2006 through 2009. Downloaded January 23rd from: http://www.gop.gov/Committeecentral/bills/hr3402.asp

Gordon, S. (2005).Exploring Spyware and Adware Risk Assessment. Presentation to the Computers and Security Institute Conference. Phoenix, Arizona.

Gordon, S. (2005a). Changing the Way the World Thinks about Computer Security. Ph.D. thesis. University of Middlesex. Department of Computer Science. London, UK.

Gordon S., & Ford R. (2004). Cyberterrorism?, in Cybterterrorism, The International Library of Essays in Terrorism, Alan O'Day, Ashgate, ISBN 0 7546 2426 9 (2004)

Helpguide (2005). Domestic Violence and Abuse: Types, Signs, Symptoms, Causes, and Effects. A project of the Rotary Club of Santa Monica, California and the Center for Healthy Again. Downloaded January 23rd 2006 from: http://www.helpguide.org/mental/domestic_violence_abuse_types_signs_causes_effects.htm

Informit, (2005). Protecting Yourself From Internet Crime, Part II. Downloaded Jan 23rd 2006 from: http://www.informit.com/guides/content.asp?g=security&seqNum=144

ISTR (2005). Symantec Internet Security Threat Report. Trends for July 04- December 04 Volume VII. March, 2005.

Kuhn T. (1992). The Structure of Scientific Revolution. 2nd edition, ed. University of Chicago Press. Chicago, Illinois.

Krone T. (2005). High Tech Crime Brief. Australian Institute of Criminology. Canberra, Australia. ISSN 1832-3413.

Parker, D. (1998). Fighting Computer Crime: A New Framework for Protecting Information ISBN 0471163783. John Wiley & Sons Inc.

PC Magazine (2005). PC Magazine Encyclopedia. Downloaded Jan 23rd 2006 from: http://www.pcmag.com/encyclopedia_term/0,2542,t=crimeware&i=55434,00.asp

Studycrime (2005). Online Resource Guide to Law and Crime. Downloaded Jan 23rd 2006 from: http://www.studycrime.com/Crime/Crimeware.php

United Nations (1995). The United Nations Manual on the Prevention and Control of Computer Related Crime , 1995. supra note 41, paragraphs 20 to 73 in International Review of Criminal Policy, 43-44. 1995.

Wikipedia (2006). Common Usage Crimeware Definition. Downloaded Jan 23rd 2006 from http://en.wikipedia.org/wiki/Crimeware.

Zeviar-Geese (1997-98). The State of the Law on Cyberjurisdiction and Cybercrime on the Internet. California Pacific School of Law. Gonzaga Journal of International Law. Volume 1. 1997-1998.

ZD (2005). 2.8 Billion in E-Commerce Revenues lost to Fraud in 2005. ZD Online., 2005. Downloaded Jan 23rd 2006 from: http://blogs.zdnet.com/ITFacts/?p=9471

Zona, M.A., Palarea, R.E., & Lane, J.C. (1998). Psychiatric Diagnosis and the Offender-Victim Typology of Stalking. In Meloy, J.R. (1998). The Psychology of Stalking: Clinical and Forensic Perspectives. San Diego, California: Academic Press

Zona, M.A., Sharma, K.K., & Lane, M.D. (1993). A Comparative Study of Erotomanic and Obsessional Subjects in a Forensic Sample. Journal of Forensic Sciences, 38, p. 894 – 903

End-To-End Security Implementation For Mobile Devices Using TLS Protocol

Barış Kayayurt & Tugkan Tuglular
Department Of Computer Engineering, Izmir Institute of Technology, Turkey

About Author(s)

Barış Kayayurt is a Software Developer, M.Sc.
Contact Details: Havelsan, Eskisehir Yolu 7. Km., 06520 Ankara, Turkey, phone +90-535-4239314, e-mail kayayurt@bornova.ege.edu.tr

Tugkan Tuglular is an Assistant Professor, Ph.D.
Contact Details: Department of Computer Engineering, Izmir Institute of Technology, Gulbahce Koyu, Urla, Izmir, Turkey, phone +90-232-7506550, fax +90-232-7506562, e-mail tugkantuglular@iyte.edu.tr

Keywords

Wireless Security, Mobile Communication Security, End-to-End Security, SSL, TLS, Cryptographic Functions, Mobile Communication Architecture, Wireless Applications, J2ME, Mobile Devices

End-To-End Security Implementation For Mobile Devices Using TLS Protocol

Abstract

End-to-end security has been an emerging need for mobile devices with the widespread use of personal digital assistants and mobile phones. Transport Layer Security Protocol (TLS) is an end-to-end security protocol that is commonly used on Internet, together with its predecessor, SSL protocol. By implementing TLS protocol in mobile world, the advantage of the proven security model of this protocol can be utilized. The main design goals of mobile end-to-end security protocol are maintainability and extensibility. Cryptographic operations are performed with a free library, Bouncy Castle Cryptography Package. The object oriented architecture of proposed end-to-end security protocol implementation makes the replacement of this library with another cryptography package easier. The implementation has been experimented with different cases, which represent use of different cryptographic algorithms.

Introduction

Current trend in mobile applications have been the enterprise-style applications that needed high-capacity, network-connected devices. Financial applications like banking and stock trading are common examples of these kinds of applications. However, some deficiencies prevent its acceptance in e-commerce applications (Soriano & Ponce, 2002).With more and more network connected applications, security has became one of the most popular concept in mobile community. Mobile security deals mainly with two issues: security of the physical device and its contents and security of the data in network communication.

The communication of data in mobile devices is provided by the mobile networks. Mobile networks are open to many kinds of attacks. The open data communication in these networks may cause attacks against the secrecy, integrity and authenticity of data. Many vendors have proposed solutions against these security vulnerabilities. Most of these solutions are vendor-specific proprietary products or libraries.

Jøsang and Sanderud (2003) investigated the aspects of mobile networks and found that it is both harder and easier to implement communication security as compared to for example the Internet. They concluded that communication between mobile and fixed networks create particular problems regarding security protocol design. Data security problems can be minimized through the implementation of end-to-end security in this proxy-based environment.

Many mobile networks have proxy-based architecture. In this architecture, security between the mobile device and the proxy server is provided with a solution and the security between the proxy server and the destination server is provided with another solution. WTLS, announced as the security solution of WAP protocol is such a protocol. There are two problems with proxy-based solutions. First, it decreases performance. As the data is decoded and reencoded at proxy server, it may cause latency. Second, regardless of using WTLS, a malicious operator can eavesdrop and tamper with the data (Mynttinen, 2000). These attacks may threat the data security between the period it is decoded and encoded.

The alternative to proxy based security solutions is end-to-end security. End-to-end security refers to the securely encoding of data at the source host and decoding at the destination host. The data will not travel unencoded at any part of the communication. TLS (Transport Layer Security Protocol) and its predecessor SSL (Secure Sockets Layer) protocols are end-to-end security

solutions that are commonly used in wired world (Agarwal et al., 2004). There are a number of reasons to use these protocols in the mobile world:

- TLS and SSL protocols have been used for many years, so the security of them are tested by the community and accepted as secure enough to be used by financial data communication.
- TLS and SSL protocols are common in wired world and may be accepted as the security protocol of Internet. Using these in mobile applications will make the integration between mobile applications and Internet easier.
- TLS and SSL have open specifications and may implementations. It may be relatively easy to implement these for specific needs.

Both protocols have known by their resource consuming nature and thought of not being suitable for mobile networks. The end-to-end security solution presented in this paper is implemented for mobile devices using Java platform in order to show that it is possible and efficient to use TLS in mobile networks. Kwon et al. (2001) worked on the same problem and published an end-to-end security model that utilizes transport layer security to overcome WAP security problem. However, implementation of their model and related experiments with resulting values cannot be found.

In this paper, only transport layer security for mobile devices has been discussed. Other security issues are out of scope of this paper. Any implementer of the transport layer security for mobile devices should be aware of the security weaknesses that the J2ME environment contains (Healy & Filiol, 2005).

This paper presents a solution to end-to-end security needs of mobile devices, such as PDAs and mobile phones. After brief explanation of mobile architectures is given, the proposed architecture for end-to-end security of mobile devices is described. Next section mentions the implementation details of the architecture; the development environment and some implementation issues. Then the experiment enviroment and experiments are explained together with their results. The paper ends with a discussion of the results of experiments.

Mobile Architectures

Mobile Java applications run on mobile devices, mostly mobile phones and PDA (Personal Digital Assistant) devices. Whatever the device is, the application platform has a unique architecture. This architecture consists of the device's architecture (hardware, OS, JVM, etc.), network connection architecture that connects the mobile device to the outside world and the application architecture. The mobile security protocol developed will run on the architecture explained below.

Device Architecture

Today, many cell phones and PDAs support MIDP (Mobile Information Device Profile) technology. Usually, cell phones have built-in support for either MIDP 1.0 or 2.0; and PDAs (mostly PALM based PDAs) support MIDP after installing the KVM (Kilobyte Virtual Machine) specific for these devices. A variety of devices from different vendors have J2ME (Java 2 Micro Edition) MIDP support (Knudsen, 2001). In order to support this kind of flexibility and customization need, J2ME architecture is designed to be modular and scalable. Figure 1 shows the general architecture of a J2ME MIDP device with its layered approach, based on (Sun Microsystems, 2000a).

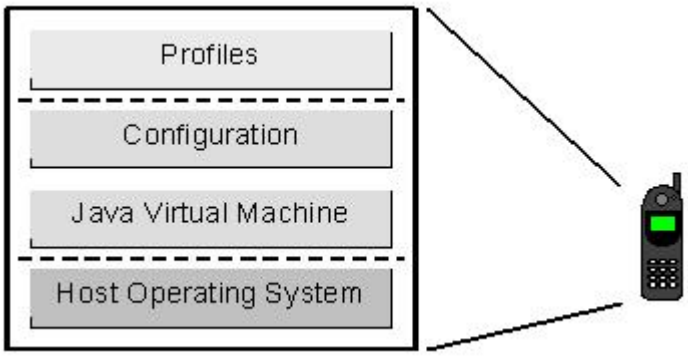


Figure 1: J2ME Device Architecture (Sun Microsystems, 2000a)

In this architecture, the host operating system communicates with the device hardware and provides services for the Java Virtual Machine layer. JVM layer is the implementation of a Java virtual machine that is customized for a particular device's host operating system and supports a particular J2ME configuration. The configuration layer defines the minimum set of Java virtual machine features and Java class libraries available on a particular category of devices. The profile layer defines the minimum set of Application Programming Interfaces (APIs) available on a particular family of devices. The layer where Java applications are written is the Profile Layer.

Mobile device operating systems control the mobile device hardware and provide services for the Java Virtual Machine Layer. Although MIDP applications run on top of a JVM and is independent of the operating system, the implementation of the JVM is customized for the operating system. So the operating systems capabilities define the MIDP application capabilities. The most common operating systems in MIDP supporting devices are PALM OS and Symbian OS.

As in desktop, Java applications on mobile devices run on top of a virtual machine called Java Virtual Machine (JVM). The virtual machine is implemented in native code and may directly use operating system services. The VM of mobile devices are different from J2SE JVM because of the limited resources and device architectures of mobile devices. The K Virtual Machine (KVM) is a highly portable JVM designed from the ground up for small memory, limited-resource and network-connected devices such as cellular phones, pagers, and personal organizers (Sun Microsystems, 2000a).

Configuration Layer defines the minimum set of Java virtual machine features and Java class libraries available on a particular "category" of devices. Connected, Limited Configuration Layer (CLDC) is the configuration specified for small, resource constraint mobile devices. CLDC specification defines Java language and virtual machine features, core libraries, input/output, networking, security and internationalization (Sun Microsystems, 2000b). CLDC configuration does not address application life-cycle management (installation, launching, deletion), user interface, event handling and high level application model. These features are addressed by profiles implemented on top of the CLDC.

Profile Layer defines the minimum set of Application Programming Interfaces (APIs) available on a particular family of devices. Profiles are implemented on top of configurations. A device can support multiple profiles. MIDP (see (Sun Microsystems, 2000c) and (Sun Microsystems, 2002)) is the most common profile implemented in all J2ME profiles. MIDP defines a very limited API because of size and performance reasons. However, it can be extended with various optional packages. Bluetooth (JSR 82), Web services (JSR 172), wireless messaging (JSR 120), multimedia (JSR 135), and database connectivity are some of the optional packages developed for CLDC /

MIDP environments. Device manufacturers may or may not include optional packages in their device implementations.

Network Connection Architecture

A Wireless Local Area Network (WLAN) is a flexible data communications system that can either replace or extend a wired LAN to provide added functionality (Intel, 2000). WLANs use Radio Frequency (RF) to transmit and receive data over the air without cables. Data is superimposed onto a radio wave through a process called modulation, and this carrier wave then acts as the transmission medium, taking the place of a wire. WLANs use the 2.4 Gigahertz (GHz) frequency band. WLANs offer all the features of traditional Ethernet or Token Ring networks with the addition of increased network infrastructure flexibility. This flexibility makes wireless networks an important alternative to wired-networks in small areas. WLANs provide a communication medium for wireless mobile devices (PALM, e.g.) in small areas like a building.

An alternatively new technology in mobile device communication is Bluetooth. Bluetooth provides short-range wireless connectivity over radio-frequency technology that uses the 2.4 GHz Industrial-Scientific-Medical (ISM) band (Mahmoud, 2003). Bluetooth lets mobile devices to communicate with each other up to 10 meters range and 1 Mbit/sec speed. The IEEE has designated its version of Bluetooth with 20Mbit/sec speed, as the IEEE 802.15 standard.

GPRS (General Packet Radio Service) is an enhancement of core GSM (Global System for Mobile Communications) networks that allows the rapid transfer of data bundled into packets, separate from voice or data call circuits (Symbian, 2001). GPRS is a 2.5 Generation (2.5G) technology that is the last stone before the coming 3G networks that will give high speed access allowing live video. GPRS data is transmitted in packets, up to the 20 or 30 Kbps speed though the theoretical maximum speed is 171.2 Kbps. GPRS set-up time is short and connection is always on.

Application Architecture

Client/server architecture is the main architecture for mobile network applications. In this architecture, a network-aware application residing on a wireless device, client, connects with back-end applications and servers behind a firewall or proxy gateway over a wireless network and Internet and corresponding communication protocols. Figure 2 shows the client/server architecture of a mobile application.

In Figure 2, the wireless devices can be mobile phones, PDAs or two-way pagers. They have the mobile applications on top of MIDP and communicate with an antenna over a wireless communication protocol. These protocols can be Wireless-LAN on a small area or GPRS on a wide area. The antenna is directly connected to a wired network like Internet, which connects the mobile device to the back-end server systems.

Mobile applications typically use HTTP as the application-level communication protocol as it is common and can pass firewalls. HTTP is the mandatory protocol in MIDP 1.0 (Sun Microsystems, 2000c) and MIDP 2.0 specifications (Sun Microsystems, 2002). With MIDP 2.0, low-level socket APIs can also be used to directly communicate with the server application below the application level.

TLS may provide an application-level end-to-end security on top of sockets in this architecture. The TLS implementation developed in this thesis runs on top of pure sockets and provides an application level security between a MIDP application and server back-end application. The MIDP version of the developed TLS protocol communicates with the J2SE version of the protocol and

sends object data over the secure connection. The mobile device always runs the client version of TLS protocol and back-end server always runs the server version of the TLS protocol.

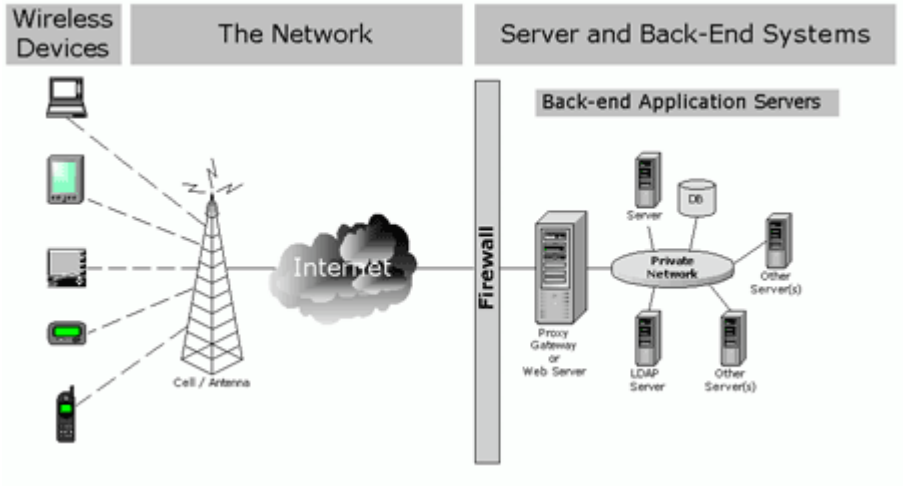


Figure 2: Environment Of A Typical Networked Wireless Application (Ortiz, 2003)

Peer To Peer Architecture is an alternative and new architecture for wireless devices. In this architecture, the two devices communicate directly with each other. This communication may be the direct communication of the devices over a communication medium like Bluetooth or it may be an application level communication with the two mobile applications communicating over a central server.

The client side of the application is same with Client/Server Architecture. It uses either HTTP or low-level socket APIs, either stream-based or datagram-based, for network communication. The difference is in server side. In this architecture, the server is also a mobile device. MIDP 2.0 has an API for server socket that may be used for this purpose.

The TLS implementation to work in this architecture needs both client and server versions of TLS to work in the mobile device. This may be impractical for two reasons: First, resource constraints of mobile devices may cause performance problems for server side implementation of the TLS protocol. Second, server sockets are not commonly implemented in MIDP devices. In spite of these deficiencies, the TLS implementation presented in this paper can be used in peer-to-peer architectures for experimental purposes.

Design of Mobile Devices End-To-End Security Architecture

The security protocol presented in this paper is an end-to-end security protocol based on TLS 1.0 specifications and adopted to work on J2ME mobile devices as well as standard Java VMs. The protocol implementation itself is also an application although high-level applications may use it through its APIs to transmit data and objects securely.

The security protocol architecture developed covers both TLS protocol architecture and the necessary APIs architecture. TLS protocol architecture is based on TLS 1.0 specification and has some additions and subtractions. The necessary APIs architecture involves cryptography model classes that abstract the real implementations of cryptography packages; socket classes that abstract TCP or UDP based socket implementations and the serialization of object data; XML Serializer architecture that is a standalone API incorporated into the protocol implementation. The main design issues taken into consideration during the definition of the architecture of protocol

implementation are J2ME compatibility, mobile adaptation, secure object transmission, full abstraction and complete solution.

The main architecture of the mobile end-to-end security protocol is based on the TLS 1.0 protocol specification (Dierks & Allen, 1999). TLS is a protocol that consists of several layers of protocols as shown in Figure 3. At the bottom of these layers, there is the TLS Record Protocol. The Record Protocol is responsible for taking messages to be transmitted, fragmenting into blocks, optionally compressing, applying MAC to protect data integrity, encrypting the block and transmitting the result to higher level clients. Received data is then decrypted; applied MAC is verified to protect data integrity; optionally decompressed; reassembled and then delivered to higher level clients by the Record Protocol. TLS Record Protocol blocks are transported over a reliable transport protocol like TCP.

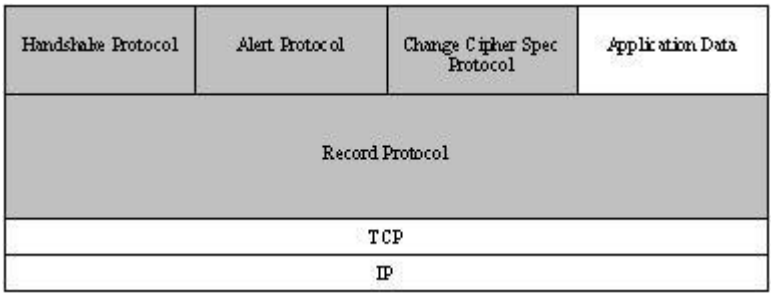


Figure 3: The Layered Structure Of TLS Protocol

There are four clients of record protocol: TLS Handshake Protocol, TLS Alert Protocol, TLS Change Cipher Spec Protocol and application data. TLS 1.0 Specification allows new record protocol clients to be supported by the record protocol.

TLS Handshake Protocol is used to allow peers to authenticate each other and to exchange cryptographic parameters that are used in TLS Protocol. TLS Handshake is performed in the beginning of a session before the application data is transmitted or received. The Change Cipher Spec Protocol is used to start to use new keys and encryption methods that the Handshake Protocol has established. TLS Alert Protocol is used to send warning and fatal level errors that could occur in TLS session. After the handshake is performed, application data is taken by Record Layer and sent securely to the other peer.

In our model, the Record Layer behavior is implemented in the class RecordLayerImpl and the Handshake Layer behavior is implemented in the class HandshakeLayerImpl. This class has two instances of class RecordLayerImpl; one for current state and one for pending state. Both HandshakeLayerImpl and RecordLayerImpl classes are transparent of the underlying socket implementations and talk to the peer classes in both side of the communication.

The classes TLSClientSocket and TLSServerSocketListener are the only classes needed to be known by the applications that will use this implementation of the protocol. An application that needs to be the client in the secure communication must use the class TLSClientSocket; and an application that needs to be the server in the secure communication must use the class TLSServerSocketListener. The class TLSServerSocket is used to handle a secure connection session in the server-side as a separate thread. An application may use the class TLSSocketFactory to obtain either a TLSClientSocket class instance or a TLSServerSocketListener class instance.

The mobile secure application is designed as the protocol implementation separated from the underlying communication model. This is achieved by the interface TLSSocketImpl. This interface defines the operations needed for sending and receiving of secure data and implemented by the

client and server socket classes that abstract the communication details from the protocol details. Thus, the communication of data can be changed without changing any protocol dependent code. The class TLSTCPSocketImpl is the client-side implementation of the interface TLSSocketImpl as TCP based stream socket and the class TLSTCPServerSocketImpl is the server-side implementation of the interface TLSSocketImpl as TCP based stream server socket. The class TLSUDPSocketImpl is the client-side implementation of the interface TLSSocketImpl as UDP based datagram socket and the class TLSUDPServerSocketImpl is the server-side implementation of the interface TLSSocketImpl as UDP based datagram socket. Figure 4 shows the main object model of the mobile end-to-end security protocol.

The class HandshakeLayerImpl defines operations and attributes needed to perform Handshake Layer behavior and to send and receive application-level data after the establishment of a secure session. Each secure session has one instance of class HandshakeLayerImpl and the class instances at both side of the communication talk to each other. As the Handshake Layer in TLS Protocol communicates with the Record Layer to send and receive data, the HandshakeLayerImpl class uses RecordLayerImpl class to send and receive data.

The class RecordLayerImpl defines operations and attributes needed to perform Record Layer behavior. The responsibilities of RecordLayerImpl class are to send and receive application and Handshake Layer data after encoding and decoding; generate new TLS keys according to the exchanged security parameters; copy security parameters from pending state to current state and activate new TLS keys for read side and write side.

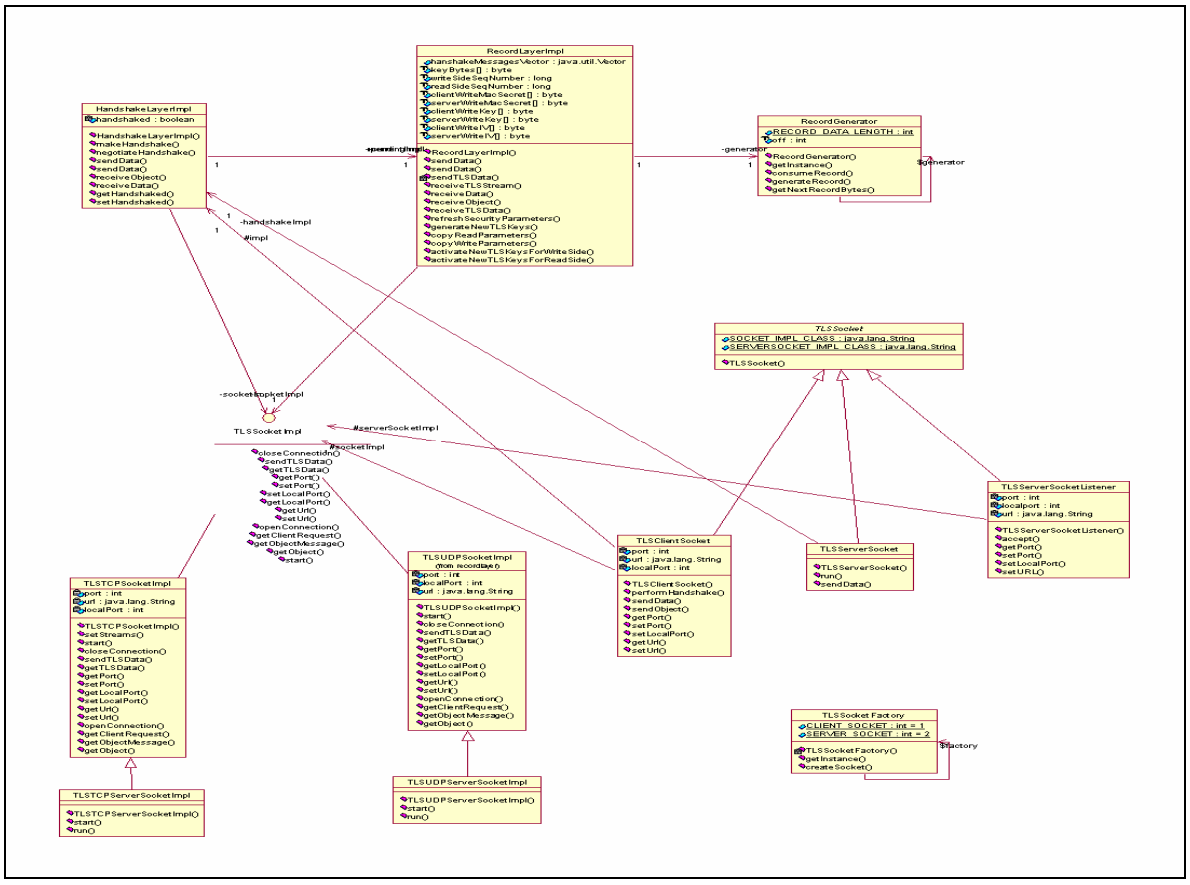


Figure 4: Main Object Model Of The Mobile End-To-End Security Protocol

In TLS 1.0, the Record Layer performs encryption, MAC and compression functions according to the algorithms exchanged during handshake procedure. The Record Layer architecture of the mobile protocol is modeled to support this need in an extensible way. This achieved with the use of the interfaces TLSEncryption, TLSMac and TLSCompression. These interfaces define the base methods to perform encryption, decryption, adding MAC, verifying MAC, compression and decompression. Algorithms are implemented in the classes implementing these interfaces. This method eases the addition of a new algorithm to the protocol implementation.

Implementation of Mobile Devices End-To-End Security Architecture

The mobile end-to-end security protocol is implemented in Java to be compliant to both J2ME and J2SE platforms. The development environment for mobile end-to-end security protocol is JBuilder 8.0 Enterprise MobileSet 3 Integrated Development Environment. The development environment was chosen because it has an integrated MIDP application development support. A project may be compiled with MIDP libraries by changing the project JDK to J2MEWTK from project properties window. JBuilder IDE was also chosen for its high performance, code completion feature and user-friendly interface.

JBuilder IDE uses Sun Microsystem's J2ME Wireless Toolkit for MIDP development. J2ME Wireless Toolkit (J2MEWTK) is a toolkit that can be used separate or plugged into a development IDE. It provides development tools and emulators for MIDP application development. The 2.1 version of J2MEWTK is used in the development and in the experiements of the developed protocol for mobile environments. J2MEWTK 2.1 supports MIDP 2.0, CLDC 1.1, optional Wireless Messaging API (JSR 120), Mobile Media API (JSR 135) and Web Services Access for J2ME API (JSR 172).

J2ME Wireless Toolkit has the tool Ktoolbar that manages MIDP application development. Ktoolbar has menu options for creating a new MIDP application, opening an existing one, compiling and running the application with the set up MIDP version, changing the J2MEWTK and application preferences.

J2MEWTK uses device emulators to run MIDP applications. Device emulators are software implementations of mobile devices to test applications before the real deployment. Device emulators provide a faster and easier development for mobile applications. Most mobile phones and PALM devices have their emulator software. Mobile phone emulators are distributed by phone vendors freely. They have the same operating system and visual interface with the original phone. PALM emulators emulate PALM OS. A ROM file is taken from a real PALM device with Hotsync connection and loaded into the emulator software. This gives all the features of the PALM device to the emulator software including program installation and uninstallation. Some examples are also run on original PALM M100 PDA and Nokia™ 6600 mobile phone.

Java has been divided into three platforms: J2EE for server side development, J2SE for standard desktop applications and J2ME for small, embedded devices. J2ME has its own configurations and profiles. Each configuration and profile has its own library APIs. The main target Java platform for the mobile security protocol is J2ME platform. The configuration is CLDC and the profile is MIDP. This fact caused CLDC / MIDP APIs to be used in the implementation of the protocol.

Most CLDC / MIDP language APIs are also valid in J2SE platform, that makes the developed protocol also compliant to this platform. However, because of the limits of mobile devices, network connection library has been changed. J2ME uses Generic Connection Framework (GCF) for all kinds of connections including network and file connections. This brings a lighter connection API

when compared with the heavy network API of J2SE platform. J2ME version of the protocol uses GCF and the J2SE version uses java.net package for network connections.

The security protocol implementation needs secure network connections to send and receive data between peers. Network connection implementations are abstracted from business logic classes and appear in socket classes that are used by other classes.

J2ME uses Generic Connection Framework (GCF) and J2SE uses java.net package network connection classes to provide network connections. The protocol implementation may use TCP and UDP sockets and server sockets for network connections between client and server. The socket connection type is determined by the constants in class TLSSocket.

Implementation of Cryptographic Services

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. The package is organised so that it contains a light-weight API suitable for use in any environment (including the newly released J2ME) with the additional infrastructure to conform the algorithms to the JCE framework (BouncyCastle, 2006). The Bouncy Castle Crypto package can be downloaded from http://www.bouncycastle.org/ for free. This software is distributed under a license based on the MIT X Consortium license.

Experiments and Discussion

The mobile end-to-end security protocol was implemented mainly to be used in mobile devices, PDAs and mobile phones e.g. The implemented protocol library is loaded on a real mobile phone to show that it can run on a resource-constraint environment and establish network connection on a real wireless network. Mobile end-to-end security protocol was designed and implemented to have a reliable and maintainable protocol implementation that would have acceptable performance results. This section mentions the experiments performed to measure the performance averages of the protocol implementation. It presents the scope of experiments, the platform experiments were performed, different experiment cases prepared and the evaluation of the experiment results. The result values of those experiments are shown as well.

Scope Of The Experiments

The mobile end-to-end security protocol experiments involve the performance runs of both the client and server modes of the protocol with different cipher suites. Experiments cover both J2SE (as server) and J2ME (as client) platforms. The secure connection will be established with different cipher suites and TCP is used as the underlying network connection protocol.

The experiments of the protocol are divided into two main sections:

- Experiment of the handshake procedure.
- Experiment of the application level object transmission and receiving.

As the handshake procedure has many steps, it is divided into different time spans. By this way, it is aimed to measure the real performance of the steps and to define bottlenecks in the protocol implementation. Some operations will only be available in specific cipher suites. Table 1 shows the operations in both server and client modes of the protocol.

Operation	Server	Client
Public key signing	✓	
Public key verification		✓
Key pair generation	✓	✓
Pre-master secret generation	✓	✓
Master secret generation	✓	✓
TLS keys generation	✓	✓
All handshake	✓	✓

Table 1: Mobile Security Protocol Operations Used In Experiments

Experiments were performed by using different TLS cipher suites. Server side of the program supports all cipher suites available to the mobile security protocol implementation. Each experimental client supports only one cipher suite and requests the handshake to be performed with that cipher suite when it sends ClientHello message to to the server. Thus, the connection will be established with that cipher suite. Table 2 shows cipher suites used in the protocol experiments. The first cipher suite in Table 2 is non-ephemeral that means asymmetric keys are not generated at run-time. The other cipher suites in Table 2 are ephemeral that means asymmetric keys are generated at run-time. Experiment results are organized to compare ephemeral cipher suite results with each other. The other comparison given is between non-ephemeral RSA and ephemeral RSA.

The experiments were performed according to the following scenario:

- A server listening to secure connection requests
- A client requesting a connection with the server
- Performance of the handshake procedure
- Sending of an example object from client to server over the secure connection established. The example object is called Person that has the private attributes id, name and surname.

Cipher Suite	Authentication & Key Exchange Algorithm	Symmetric Algorithm	Hash Algorithm
TLS_RSA_WITH_AES_CBC_SHA	RSA	AES	SHA
TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RSA_EXPORT	RC2	MD5
TLS_DHE_RSA_WITH_DES_CBC_SHA	DHE_RSA	DES	SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	ECDHE_ECDSA	AES	SHA

Table 2: Cipher Suites Used In Protocol Experiments

Mobile Device Experiment Platform Configuration

Mobile end-to-end security protocol was designed and implemented to be run on J2ME CLDC / MIDP platforms including cellular phones and PALM PDAs. The experiments with a real mobile device are performed by running the experimental client program on a mobile phone and server program on a desktop PC. The mobile phone is a Nokia™ 6600 smart phone with ARM9 104 Mhz CPU, 6 MB of storage memory, GPRS and Bluetooth connectivity and Symbian™ OS 7.0s operating system with MIDP 2.0 support. The network connection between the mobile phone and the desktop PC is provided with the GPRS connection provided by Turkcell mobile service carrier. The transport protocol used in the experiments is TCP.

Mobile device experiments were performed only on J2SE-J2ME architecture mentioned in section "Scope Of The Experiments". Experimental programs run 10 times and the highest (max), lowest (min) and average time durations are measured.

Mobile Device Experiment Results

Mobile device experiments were performed according to the configuration explained above. There was no problem in establishing communication with GPRS and all experiments succeeded. Tables 3 and Table 4 (see Appendix) show the secure communication time span durations when the experimental client program runs on a Nokia™ 6600 mobile phone and the server program runs on J2SE platform and transport protocol between is TCP. All values presented in the tables are milliseconds (ms). "0" value in a table means that the time duration is below milliseconds. N/A value in a table means that the operation is not applicable in the cipher suite.

Figure 5 shows the comparison chart of average client total handshake times of the experimental cipher suites when the client runs on a Nokia™ 6600 mobile phone and the server runs on J2SE platform and the transport protocol between is TCP.

The results of the experiments performed were shown in tables given in Appendix. The handshake procedure is divided into operations. Each operation is an important step in the handshake that was expected to take a measurable time. The experiments were repeated for a definite number for each case and the resulting average, max and min values were noted.

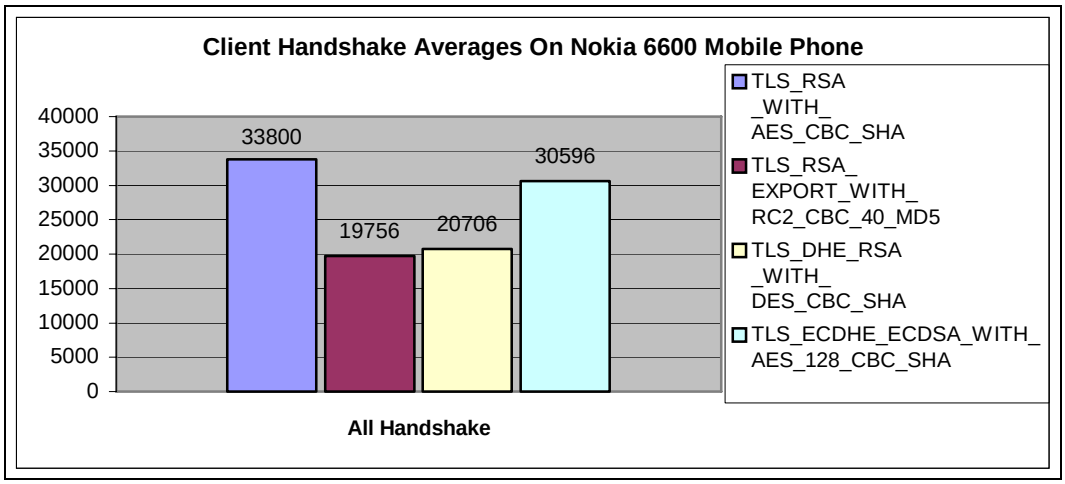


Figure 5: Client Average Handshake Times On Nokia™ 6600 Mobile Phone

Experiment results will be evaluated according to the following criteria:

- Defined Operations
- Running Platform
- Network Protocol
- Cipher Suite

All the average, min and max values increase in the J2SE-J2ME experiment results shown in Tables 3 and 4. This architecture includes a client in J2ME platform and a server in J2SE platform and expected to be widely used in real life. The lowest average total handshake value in this architecture is 19756 milliseconds of TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 cipher suite. The highest average total handshake value is 33800 milliseconds of TLS_ RSA_WITH_AES_CBC_SHA cipher suite. Considering ephemeral, the highest average total handshake value is 30596 milliseconds of TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA cipher suite. This shows that Elliptic Curve Diffie-Hellman operations have a poor performance on J2ME platform. These results are shown in the chart in Figure 5.

Conclusion

End-to-end security is an emerging need for mobile devices. Banking, military and other enterprise applications need more and more security to run on mobile devices. This project aimed at developing an extensible end-to-end security protocol implementation that could be used by both mobile and desktop applications. TLS protocol that is commonly used in Internet was chosen as the base of the developed protocol implementation.

The proposed protocol was designed and implemented. The implementation was run with different experiments and the time span of operations were measured. The protocol implementation run on a real Nokia™ 6600 mobile phone and established a secure connection with a server computer connected to the Internet over GPRS. All the experiments were successfully completed, which indicated that the protocol was properly designed and implemented with respect to specifications. The implementation guarantees the security of any transmission at most as much as TLS. The implementation did not include optional TLS specifications like client authentication, session resumption and compression.

Appendices

Time Span	TLS_ RSA_EXPORT_ WITH_ RC2_CBC_40_ MD5 (ms)			TLS_ DHE_RSA _WITH_ DES_CBC_ SHA (ms)			TLS_ ECDHE_ECDSA_ WITH_ AES_128_CBC_ SHA (ms)		
	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
Client									
Public key verification	3021	5562	2672	3271	5781	2938	8251	11156	7640
Premaster secret generation	322	359	282	17	31	0	2061	2235	1938
Master secret generation	129	281	78	120	265	63	148	250	79
Key pair generation	N/A	N/A	N/A	303	625	234	2369	2547	2219
TLS keys generation	798	4625	359	804	4610	359	841	4671	390
All Handshake	**19756**	**45125**	**16156**	**20706**	**42766**	**17000**	**30596**	**68906**	**25625**
Time Span	TLS_ RSA_EXPORT_ WITH_ RC2_CBC_40_ MD5 (ms)			TLS_ DHE_RSA _WITH_ DES_CBC_ SHA (ms)			TLS_ ECDHE_ECDSA_ WITH_ AES_128_CBC_ SHA (ms)		
	Avg.	Max	Min	Avg.	Max	Min	Avg.	Max	Min
Server									
Public key signing	43	121	20	42	90	30	72	160	40
Key pair generation	811	2673	210	1743	4767	30	N/A	N/A	N/A
Premaster secret generation	26	31	20	1	10	0	17	115	11
Master secret generation	8	60	0	8	60	0	5	50	0
TLS keys generation	10	60	0	42	421	0	10	60	0
All Handshake	**11106**	**22943**	**9243**	**12060**	**23194**	**9764**	**20364**	**32717**	**18256**

Table 3: Cryptographic Operation Durations In The Mobile Device Experiments For Ephemeral

Time Span	TLS_ RSA _WITH_ AES_CBC_ SHA (ms)			TLS_ RSA_EXPORT_ WITH_ RC2_CBC_40_ MD5 (ms)		
	Avg.	Max	Min	Avg.	Max	Min
Client						
Public key verification	N/A	N/A	N/A	3021	5562	2672
Premaster secret generation	600	1781	578	322	359	282
Master secret generation	167	312	125	129	281	78
Key pair generation	N/A	N/A	N/A	N/A	N/A	N/A
TLS keys generation	801	4234	390	798	4625	359
All Handshake	**33800**	**82375**	**13907**	**19756**	**45125**	**16156**
Time Span	**TLS_ RSA _WITH_ AES_CBC_ SHA (ms)**			**TLS_ RSA_EXPORT_ WITH_ RC2_CBC_40_ MD5 (ms)**		
	Avg.	Max	Min	Avg.	Max	Min
Server						
Public key signing	N/A	N/A	N/A	43	121	20
Key pair generation	N/A	N/A	N/A	811	2673	210
Premaster secret generation	55	120	30	26	31	20
Master secret generation	7	50	0	8	60	0
TLS keys generation	8	60	0	10	60	0
All Handshake	**19657**	**64893**	**6840**	**11106**	**22943**	**9243**

Table 4: Cryptographic Operation Durations In The Mobile Device Experiments For RSA Ciphers

References

Agarwal, A. K., Gill, J. S., & Wang, W. (2004). An Experimental Study on Wireless Security Protocols over Mobile IP Networks. IEEE Proceedings of 60th Vehicular Technology Conference, 2004.

BouncyCastle. (2006). Bouncy Castle Documentation, Retrieved 10 March, 2006, from http://www.bouncycastle.org/documentation.html

Dierks, T., & Allen, C. (1999). The TLS Protocol Version 1.0, IETF RFC 2246.

Healy, J., & Filiol, E. (2005). J2ME Low Level Security: Implementation Versus Specification. Retrieved 10 March, 2006, from http://prdownloads.sourceforge.net/tinapoc/Reynaud_J2ME.pdf?download

Intel. (2000). IEEE 802.11b High Rate Wireless Local Area Networks, Intel Corporation.

Jøsang, A., & Sanderud, G. (2003). Security in Mobile Communications: Challenges and Opportunities. Australasian Information Security Workshop (AISW2003), Australia.

Knudsen, J. (2001). Introduction to Wireless Java Technologies White Paper, Sun Microsystems Inc.

Kwon, E., Cho, Y., & Chae, K. (2001). Integrated transport layer security: end-to-end security model between WTLS and TLS. IEEE Proceedings of 15th International Conference on Information Networking, 65 – 71.

Mahmoud, Q. H. (2003). Wireless Application Programming with J2ME and Bluetooth. Retrieved 10 March, 2006, from http://developers.sun.com/techtopics/mobility/midp/articles/bluetooth1/

Mynttinen, J. (2000). End-to-end security of mobile data in GSM. Tik-110.501 Seminar on Network Security. Helsinki University of Technology.

Ortiz, E. (2003). The Complexity of Developing Mobile Networked Data Services, J2ME Wireless Connection Wizard For Sun ONE Studio. Retrieved 10 March, 2006, from http://developers.sun.com/techtopics/mobility/midp/articles/wizard/index.html

Soriano M., & Ponce, D. (2002). A Security and Usability Proposal for Mobile Electronic Commerce. IEEE Communications Magazine, August 2002, 62-67.

Sun Microsystems. (2000a). Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices White Paper, Sun Microsystems Inc.

Sun Microsystems. (2000b). Connected, Limited Device Configuration 1.0a Specification, Sun Microsystems Inc.

Sun Microsystems. (2000c). JSR-000037 Mobile Information Device Profile (MIDP) 1.0 Specification, Sun Microsystems Inc.

Sun Microsystems. (2002). JSR-000118 Mobile Information Device Profile 2.0 Specification, Sun Microsystems Inc.

Symbian. (2001). Symbian on GPRS. Retrieved 10 March, 2006, from http://www.symbian.com/technology/standard-gprs.html

How to assess the *effectiveness* of your anti-virus?

Sébastien Josse

About Author(s)

Sébastien Josse is an IT consultant in a Security Evaluation Lab and a Ph D student. His doctoral dissertation topics are symmetric encryption systems and use of cryptographic mechanisms in computer virology.

Contact Details:

Silicomp-AQL, 1 rue de la Châtaigneraie, CS 51766, Cesson-Sévigné, France, Phone +33-0-0299125000, e-mail Sebastien.Josse@aql.fr

Ecole Supérieure et d'Application des Transmissions, Laboratoire de virologie et de cryptologie , B.P. 18, 35998 Rennes, France, phone +33-2-99843609, fax +33-2-99843609, e-mail Sebastien.Josse@esat.terre.defense.gouv.fr

Keywords

Anti-virus, Common Criteria, protection profile

How to assess the *effectiveness* of your anti-virus?

Abstract

This paper presents an approach whose purpose aims at supporting and making easier and more relevant the choice of an anti-virus product.

Among qualities, which one can expect from an anti-virus product, appear classically the optimal use of the resources and the reactivity of the manufacturer, particularly concerning the viral signature base update. If these requirements are significant, other methodical and technical verifications may be required in order for an individual or a company to make their choice.

In the Common Criteria evaluation scheme, a protection profile is proposed to help a software manufacturer to design a product that should be evaluated by an independent security evaluation laboratory. Protection profiles are written in accordance with the Common Criteria standard.

Starting from a protection profile, we list some tests that could be carried out to validate the security requirements of an anti-virus product.

Both use of a protection profile and the specification of tests seems to be a valuable basis to measure the confidence to grant an anti-virus product.

Introduction

Choosing an anti-virus product may be difficult. One of the reasons is that internal mechanics of these products (in particular algorithms used and design features) are generally not well documented.

Consequently, how can one be sure to make the right choice?

Some companies grant anti-virus products with awards or quality certificates:

- VB100% Award (Virus Bulletin, 2006). To get this label, an anti-virus product must detect all viruses from a pre-set list (*In-the-Wild* viruses) and must generate no false positive when analyzing a set of clean files.

- WCL Checkmark (West Coast Labs, 2006). To get the Checkmark, level 1, an anti-virus product must detect all In-the-Wild viruses and should not cause any false alarms when analyzing a set of clean files. To get the Checkmark, level 2, the product must in addition get ride of all viruses on the WildList (which are capable of disinfection). Additional checkmarks can be provided by WCL, which assess the detection rate against other malware types (Trojan Checkmark, Spyware Checkmark).

- ICSA Labs Certified (International Computer Security Association Labs, 2006). To get the certificate, an anti-virus product must satisfy all requirements in the primary criteria module, which correspond to its product category (for example, Desktop/Server anti-virus product category). Testing against the secondary criteria module (cleaning module) is optional. *Virus Certification Test Suites Matrix* maps

anti-virus types towards several virus collections (In-the-wild, common infectors and Zoo).

Virus Bulletin, West Coast Labs, and ICSA Labs tests focus on detection rates. All these laboratories use (among additional virus collections) the well-known WildList for their tests, a list of viruses that are known to be active and widespread.

ICSA Labs own certification criteria are quite similar to the Common Criteria: anti-virus products satisfying the requirements in the primary module should have the capability to: detect and prevent from the replication of viruses, report no false positives, log the results of virus detection attempts, and perform necessary administrative functions. Those requirements can be seen as a subset of functional Common Criteria requirements that are described further in this document.

However, ICSA Lab's approach does not take into account the insurance components related to the design choice or the quality of the product development. ICSA Lab certification criteria are based neither on fundamental design, engineering principles nor on an assessment of underlying technology. Their approach is a black-box approach, which is results-oriented.

The main advantage of this approach is that test procedures are easily automated and repeatable for an anti-virus product. The main drawback is that the tests are not accurate enough to increase the evaluation assurance level, as they are not opened to interpretation and analysis. It is at the same time a quality and a limitation. For example, these criteria are not precise enough to make possible a product source code review.

Moreover, ICSA Labs certification criteria do not take the environment requirements of the target of evaluation into consideration. The scope of their analysis is limited to the product.

Finally, it remains difficult to know which tests have been carried out according to the criteria chosen by these companies.

In this paper, an approach is described for making the choice of an anti-virus product, easier and more relevant. Starting from a protection profile, we list some tests that should be carried out to validate the security requirements of any anti-virus product.

Our approach is complementary to those introduced by the laboratories previously mentioned. We focus on the robustness of security functions and mechanisms, with regards to the specific vulnerabilities of this product category. We enlarge the scope of their evaluation by taking into account system environment and internal design.

Goal: assess the *effectiveness* of an anti-virus product

The *effectiveness* of an anti-virus product rests on conformity to defined criteria, which corresponds to the state-of-the-art in the field and to an expression of need for security. A test bench must be used as a support for the analysis of security functions and mechanisms. The *effectiveness* of the anti-virus product is evaluated according to the robustness of its elementary mechanisms.

Result: a platform for anti-virus products evaluation

It appeared important to us, with regard to the current viral threat, to provide ourselves with tools to reproduce or simulate a viral attack.

Thus, several characteristic viral mechanisms were implemented. It should be noted that these viral techniques are more and more often applied (by the researchers) to the validation of new viral intrusion behavioural detection techniques.

These viral techniques were applied to the validation of security functions of the anti-virus products.

Most complete achievements on this topic are: the implementation of an obfuscation engine and the implementation of a tool for injecting *relocatable* code remotely.

In the context of sword-against-shield battle, it appeared important also to understand the constraints related to the implementation of an anti-virus product. Thus, several of the current components of an anti-virus product were specified and developed. Our most complete achievements on this point are: the implementation of an on-demand scanner of viral code and the implementation of a real-time monitor.

Finally, a family of tests (benchmark) has been specified and has been used for improving the *effectiveness* of the security functions of an anti-virus product against the viral threat.

Structure of the document

Tools that have been adapted or developed within the framework of our analyses are described at the same time as they are used for the tests.

Firstly, the analysis criteria used are presented, as well as the supporting elements of the selected tests.

Secondly, the platform and the most representative tests are presented.

Finally, we conclude with our achievements and results and provide future trends and topics to investigate.

Our approach

Analysis criteria: Common Criteria

First of all, to compare products it is important to define a clear target, with regards to the category of the product and the characteristics of its environment.

We used the scope defined by the protection profile (CCEVS, 2005).

A protection profile is an implementation-independent specification of information assurance security requirements. Protection profiles are a complete combination of security objectives, security related functional requirements, information assurance requirements, assumptions, and rationale. Protection profiles are written in accordance with the Common Criteria standard.

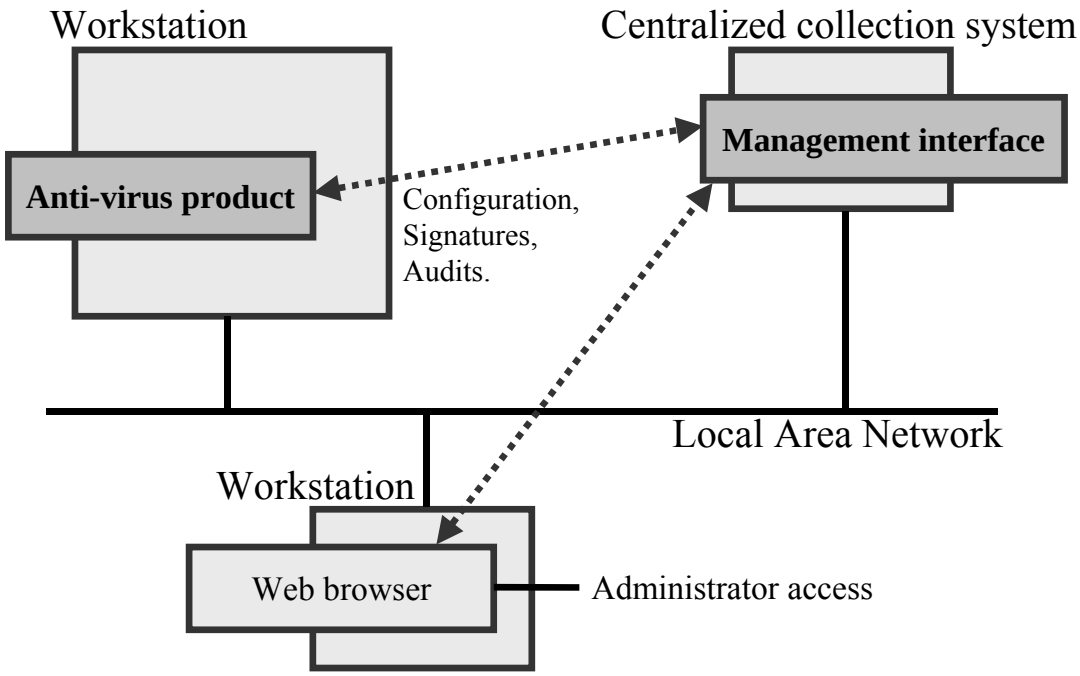
In the Common Criteria evaluation scheme, a protection profile is made to permit a software manufacturer to design a product that should be evaluated by an independent security evaluation laboratory. Security analysts perform testing and security requirements rating.

Several actors may take part in this process: security analysts, developers of the targeted anti-virus product, a third party, in charge of arbitration. For example, in France, the third party is a governmental organization, which belongs to the Prime Minister offices. The security evaluation laboratory must also be agreed by this third party.

This protection profile (CCEVS, 2005) specifies the minimum security requirements for an anti-virus application used on workstations in the US government in basic robustness environments. The target strength level (i.e. how well the target of evaluation can protect itself and its resources) of basic robustness environments is considered as sufficient for low threat environments or where compromise of protected information will not have a significant impact on mission objectives. This point is discussed in details in (CCEVS, section 3, 2005).

This protection profile is based on the Common Criteria, Version 2.2.

The following figure depicts the evaluation target and its environment.



A protection profile specifies the security requirements of an evaluation target. The evaluated target is an off-the-shelf anti-virus product. The protection profile (CCEVS, 2005) specifies these requirements relative to:

- assumptions on the environment;
- threats to which the target is exposed;
- objectives to be reached for the target and its environment;

- functional requirements to cover in order to achieve these goals;
- elements of proof concerning the way in which requirements cover the objectives; and on the way in which objectives cover the threats.

This protection profile and Common Criteria have been used as a basis (method and terminology) for our analyses. For example, in this terminology, "virus" is used generically to refer to an entire suite of exploits of security vulnerabilities, such as worms and Trojan Horses. The same term is used more specifically to refer to exploits that replicate themselves. The term "anti-virus" typically refers to measures used to counter the entire suite of exploits, not just the more specific definition of virus.

The analysis of several off-the-shelf anti-virus products, which have been chosen according to their reputation or to the statute of the manufacturer on the software market, has been carried out.

A set of tests, which were discriminatory enough to enable a comparative analysis of the products, has been gradually carried out.

Tools have been implemented to refine our understanding of the products (without reverse engineering).

Selected tests

The developed tests have been intended to validate the security functions of an anti-virus product with regard to the security requirements defined in the protection profile. These requirements relate either to the security functions of the anti-virus product, or to its environment.

Additional requirements, relating to the insurance components, make it possible to validate the quality of the software development and the technological awareness of the anti-virus product manufacturer.

At the end of this section the security requirements (functional, environment, quality) and the corresponding validation tests are given. Tests are described in the second part of this document.

The table (Appendices, table 1) points out the principal security requirements relating to the security functions of an anti-virus product. A more detailed description of these requirements can be found in (CCEVS, 2005).

The tests, which make it possible to validate these security functions (in comparison with the security requirements), are summarized in the following table. A detailed description of these tests can be found in the second part of this document.

FUNCTIONAL REQUIREMENT	**TEST**
FAV_ACT_EXP.1	**ANTI-VIRUS ACTIONS**
FAV_ACT_EXP.1.1	TST_MEM
FAV_ACT_EXP.1.2	TST_VXDAT

FAV_ACT_EXP.1.3	TST_MAIL
FAV_SCN_EXP.1	**ANTI-VIRUS SCANNING**
FAV_SCN_EXP.1.1	TST_MEM
FAV_SCN_EXP.1.2	TST_VXDAT, TST_HEURISTIC, TST_FMT, TST_DOC, TST_UNICODE
FAV_SCN_EXP.1.3	
FAV_SCN_EXP.1.4	
FAU	**SECURITY AUDIT**
FAU_GEN.1-NIAP-0347	TST_AUDIT
FAU_SAR.1	
FAU_SAR.2	
FAU_SAR.3	
FAU_STG.1-NIAP-0429	
FAU_STG.NIAP-0414-NIAP-0429	
FCS	**CRYPTOGRAPHIC SUPPORT**
FCS_COP1	TST_INTEG
FMT	**SECURITY MANAGEMENT**
FMT_MOF.1	TST_UPDATE, TST_AUDIT
FMT_MTD.1	
FMT_SMF.1	
FMT_SMR.1	
	PROTECTION OF THE SECURITY FUNCTIONS
FPT_SEP_EXP.1	TST_INSTALL, TST_INTEG

The table (Appendices, table 2) points out the main security requirements relating to the environment of the target.

The following table presents the tests, which permits the robustness of the security functions to be validated with regard to the requirements relating to the environment of the target.

ENVIRONMENT REQUIREMENT	**VALIDATION TEST**
FAU	**SECURITY AUDIT**
FAU_STG.1-NIAP-0429	TST_AUDIT
FPT	**PROTECTION OF THE SECURITY FUNCTION**

FPT_ITT.1	TST_INSTALL, TST_INTEG, TST_UPDATE
FPT_RVM.1	
FPT_SEP.1	
FPT_STM.1	

The table (Appendices, table 3) points out the main security requirements relating to the quality assurance components.

The following table presents the tests, which make it possible to check that the requirements relating to quality are covered. The latter tests series also enable us to increase our level of comprehension of the various operations that are likely to be performed by any anti-virus product.

QUALITY REQUIREMENT	**VALIDATION TEST**
ACM	**CONFIGURATION MANAGEMENT**
ACM_CAP.2	TST_INSTALL
ADO	**DELIVERY AND OPERATION**
ADO_DEL.1	TST_INSTALL
ADO_IGS.1	
ADV	**DEVELOPMENT**
ADV_FSP.1	TST_INSTALL, TST_HEURIST,
ADV_HLD.1	SRC_CODE_REVIEW
ADV_RCR.1	
AGD	**GUIDANCE DOCUMENTS**
AGD_ADM.1	TST_INSTALL
AGD_USR.1	
ALC	**LIFE CYCLE SUPPORT**
ALC_FLR.2	TST_UPDATE
ATE	**TESTS**
ATE_COV.1	TST_MAIL, TST_HEURIST, TST_MEM, TST_VXDAT, TST_AUDIT
ATE_FUN.1	
ATE_IND.2	
AVA	**VULNERABILITY ASSESSMENT**

AVA_MSU.1	TST_FMT, TST_UNICODE, TST_DOC, TST_INTEG
AVA_SOF.1	
AVA_VLA.1	

Supporting elements of the tests selected and the analysis criteria used

The problems of virus detection are, from a formal point of view, rather frozen:

- there is no system of detection/eradication which makes it possible to identify a program as being malicious without errors (Cohen, 1986) ;
- there is no algorithmic transformation which makes it possible to make the code of a malicious program semantically unintelligible (Barak, Goldreich, Impagliazzo, Rudich, Sahai, Vadhan & Yang, 2001).

As the underlying theoretical models (viral set model, obfuscator model) have today not seriously been questioned, we consider the problems of virus detection theoretically frozen.

The architecture of the detection systems must evolve and progress, from the simple pattern search engine to an expert system[1], which formulate its diagnosis (i.e. to answer the question: is this program viral?) only after having correlated several detection models (Eskin, Schultz, Stolfo & Zadok, 2001).

The evaluation of security products dedicated to viral detection must be adapted to this context (theoretical and technological):

- constant technological monitoring must be maintained, concerning:
    - the viral threat and its development;
    - anti-virus product bypassing modes :
        - propagation vectors;
        - stealth techniques;
        - protection techniques;
        - polymorphism;
    - new antiviral techniques;

[1] An expert system is able to carry out a diagnosis by comparing information, which results from one system with *a priori* knowledge available on this system (as a human expert would do). An expert system is made up at least of a knowledge base, a facts and rules base, and of an inference engine. The inference engine chooses the rules according to the observed facts.

- the specified tests must take into account:
    - the new viruses and the algorithmic innovations (Dagon, Kolesnikov & Lee, 2005) ;
    - the corrections of the published vulnerabilities;
    - the new antiviral techniques;
- the specified tests must also report:
    - the performance of the selected and implemented algorithms by the anti-virus product (naive, worked out, complexities);
    - the resistance to the black box analysis (security of implementation) (Filiol, 2006).

Both use of a protection profile (validated by the methodological approach of the Common Criteria) and the specification of tests (which give an account of the cover of the security requirements) seems to be a valuable basis to measure the confidence to grant an anti-virus product.

Description of the test platform

In this part, we present the platform and the most relevant tests.

The platform

We have positioned our anti-virus tests in relation to a fixed attack context:

- the techniques were studied on a PC Intel/M.S. Windows platform, because of its widespread distribution;
- however, the majority of the tests produced for this platform can be transposed without difficulty to other platforms.

The test platform must comprise:

- virtualisation software, like VMware (VMware, 2006);
- network probe tools;
- system diagnostic tools;
- various tools possibly dedicated to static and dynamic code analysis.

Regarding the Workstation environment parameter setting, the operating system must be re-installed and updated. No other antivirus must be installed.

Tests

We describe in this part the tests set which makes it possible to validate the security requirements relating to the security functions of an anti-virus product, its environment and the components of insurance relating to its design.

The following table presents a summary of the tests, which are described in a more detailed way in the rest of the document.

TEST	DESCRIPTION
TST_INSTALL	Installation of the anti-virus product (recovery of information on the installation and test of its robustness)
TST_HEURISTIC	Heuristic engine (recovery of information on the heuristic engine and its characteristics)
TST_MEM	Read-write memory analysis / worm blocking (validation of the ability to check memory and to protect the host against virus propagation vectors)
TST_FMT	Management of the various file formats (validation of the ability to manage various file formats, in particular when these formats are corrupted intentionally)
TST_DOC	Filtering of the document viruses (validation of the ability to detect embedded viral code of an Office or HTML document)
TST_MAIL	Analysis of incoming and outgoing mails (functional validation of the ability to analyze SMTP traffic and emails attached files)
TST_VXDAT	Viral base analysis (validation of the completeness of the viral signature base)
TST_UNICODE	Unicode support (validation of the correct implementation of Unicode format)
TST_AUDIT	Audit files (functional validation)
TST_INTEG	Integrity of the viral signature base (validation of the robustness of the viral signature base)
TST_UPDATE	Product updating (recovery of information on the product updating process)

TST_INSTALL (Installation of the anti-virus product)

This test table is divided into two parts:

- a part dedicated to recovery of information on the installation of the anti-virus product (interactions with the API and NT register). These tests are carried out during the installation;

- a part which makes it possible to test the robustness of the installation and its robustness against low level viral attacks (file system filter drivers, for example). These tests are carried out after the installation.

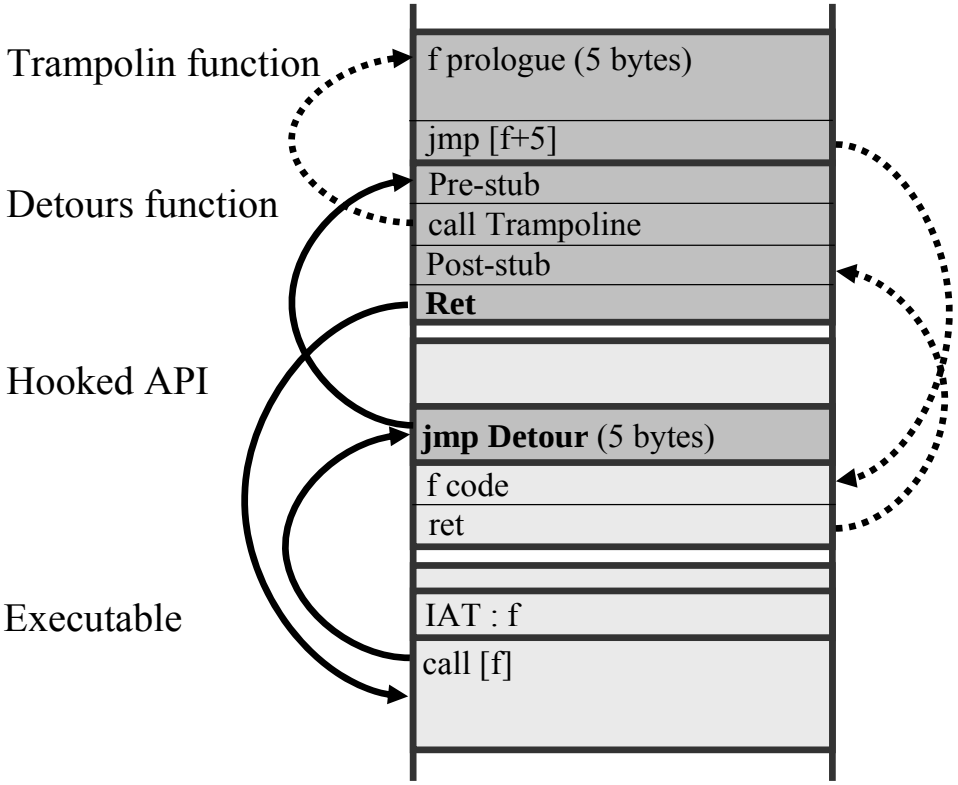
We automated the recovery of useful information at the time of the installation of the product by implementing techniques of listening to and bypassing calls according to the API functions.

There are many methods that enable the code injection into the system environment or into a process context, as well as the interception and the redirection of API Win32 function calls towards the injected code (Brubacher & Hunt, 1999; Detoisien & Dotan, 2003). Trojan horses and spyware very often implement these furtive techniques.

These techniques are also sometimes useful for tracing system calls (in particular I/O towards the file system and the register base) made by a viral program or to make certain protection software programs inoperative. These protections have been implemented for example by armoured viruses (Filiol, 2005).

It should be noted that these techniques of instrumentation and interception of calls to API Win32 functions are also applied to the behavioural detection of viral intrusion – cf. DOME, Detection of Malicious Executable (Cunningham, Khazan, Lewandowski & Rabek, 2003), and even to the implementation of real-time control engines (Winpooch, 2006).

We use some of these techniques to automatically trace system calls carried out during installation of the anti-virus product, in particular those taking part in the installation of filtering drivers (use of the detours tools (Brubacher & Hunt, 1999)).



We have also evaluated the robustness of the installation by testing (enable/unload or uninstall) the filter drivers of the anti-virus product (using fltmc and devcon tools (WinDDK, 2006; IFSKit, 2006)).

TST_HEURISTIC (Heuristic engine)

The objective of this test is to validate the correct operation of the heuristic engine of a given anti-virus product. This test also enables us to better understand the actions performed by the heuristic engine.

We consider a S_0 population of healthy programs of the system.

We generate a $S_1=T(S_0)$ population of healthy programs (Win32 executable files) having properties generally considered as suspect by any anti-virus software regarding to:

- the structure of the executable file (header, sections of data and code);
- the executable file's section code.

The *T* transformation was firstly implemented by using an adapted version of Y0da's packer (Y0da, 2006). This transformation is actually performed by the obfuscator described below, because its modularity is better.

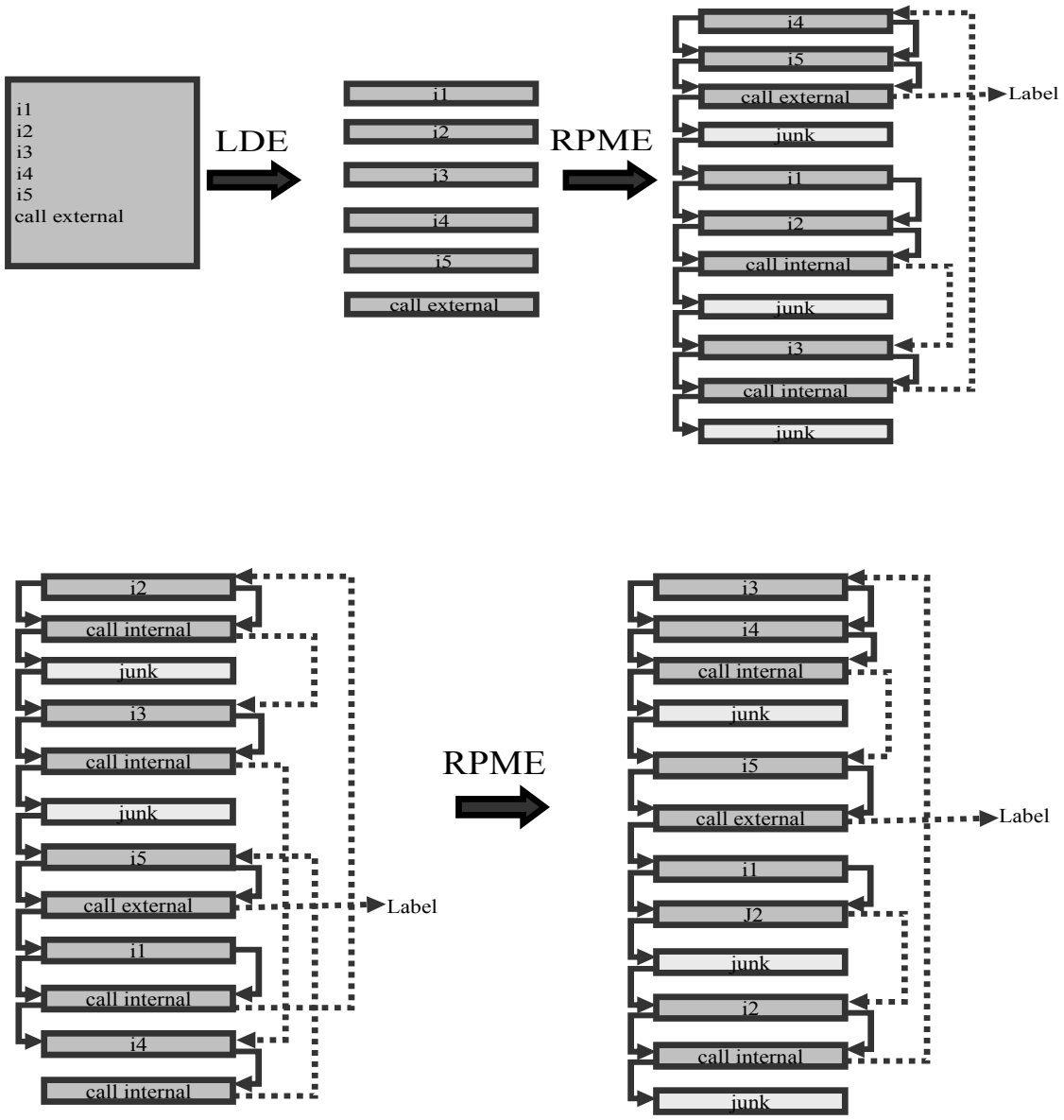
The executable files generated by the T transformation have characteristics incompatible with the code usually generated by a compiler of high-level language:

- sections are not correct (for example the last section is executable or the first section is accessible in writing, sections are incorrectly aligned or with strange names);
- header is not correct (incorrect values in the header, unusual entry point).

We consider a V_0 population made up of Win32 viruses.

We build a population $V_1=O(V_0)$ by application of an obfuscator *O*. We proceeded, as for the validation of the SAFE (Static Analyser For Executable) viral detection engine (Christodorescu & Jha, 2003), with an adaptation of the Mistfall engine (Z0mbie, 2001a, 2001b). Let us note that this idea was also used to validate the *effectiveness* of the SAVE viral detection engine (Chavez, Mukkamala, Sung & Xu, 2004), even if the authors did not automate the application of the obfuscation transformations (the transformations were applied manually, using a simple hexadecimal editor).

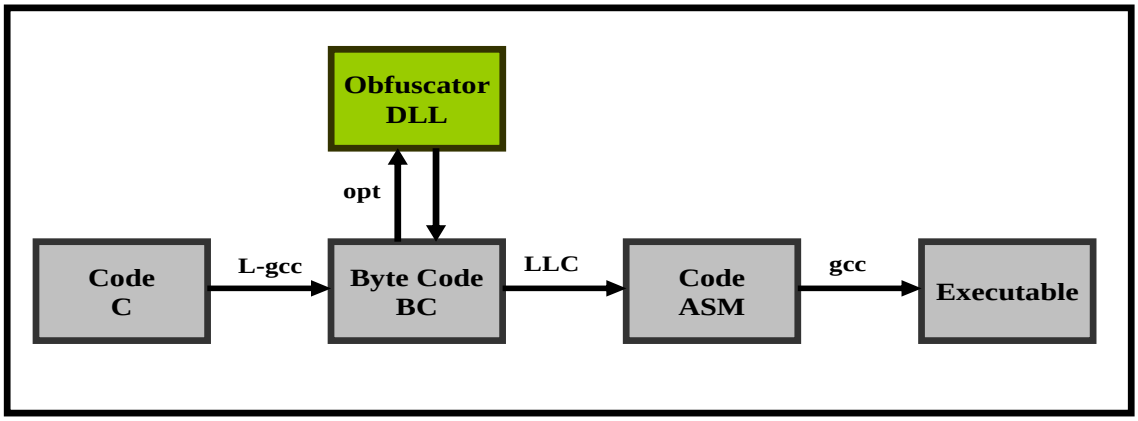
The viral engine Mistfall implements a disassembler (LDE) and a permutation engine (RPME). We instrumented this engine so that it carries out more complex obfuscation transformations (Josse, 2005). We coupled it with a generator of junk code and a base of opaque predicates.



We did not explore the possibilities offered by the generators of viral code (Virus Generator Kits (Szor, 2005)). Their use could supplement our analysis (generation *ex nihilo* of a viral population V_2.

Neither did we explore the possibilities offered by the fact of having the source code of certain viruses. We could, starting from these programs, build a V_3 population by application of an obfuscator *O* like Cloakware for the languages C/C++ and Java (Cloakware, 2006) or SandMark for the Java language (SandMark, 2006).

For example, the LLVM compilation chain (Low Level Virtual Machine, 2006) seems well suited to the application of obfuscation transformations on C source code.



Application of obfuscation transformations by using the LLVM compilation chain

The correlated analysis of the S_1, V_0 and V_1 populations makes it possible to draw several conclusions on the anti-virus product:

- its capacity to analyze polymorphic viruses;
- its capacity to analyze EPO (*Entry-Point Obscuring*) viruses (GriYo, 2006).

It is also possible to see which are the heuristic ones used, as well as the limits imposed on the level of the parameter setting by the manufacturer of the anti-virus product (depth of the analysis, etc).

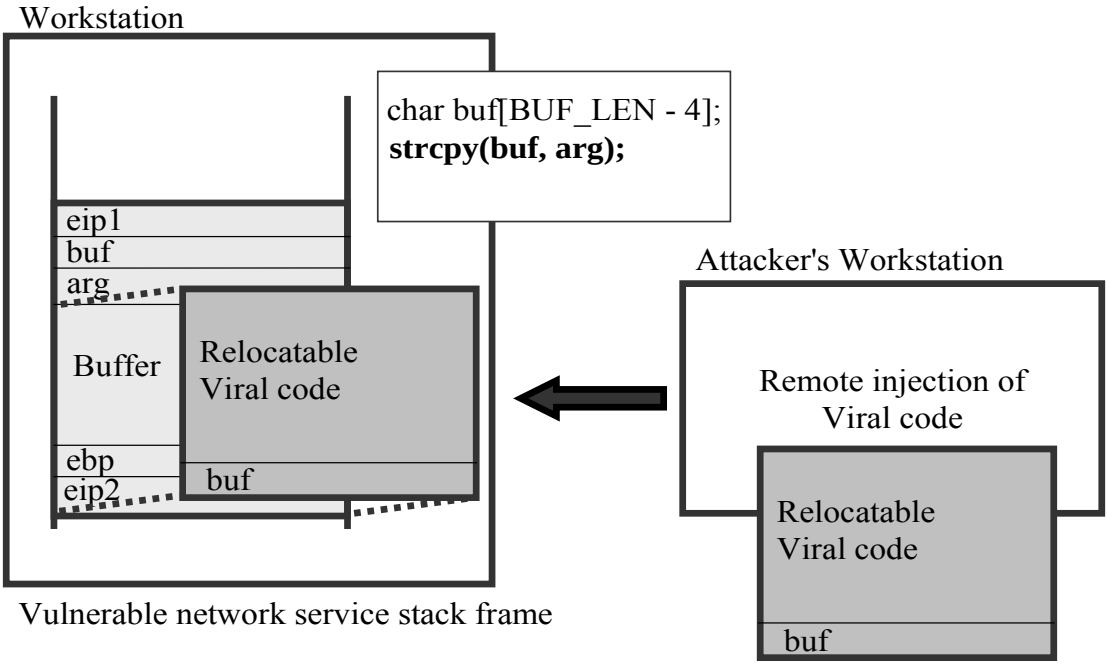
TST_MEM (Read-write memory analysis / worm blocking)

This test makes it possible to check that a virus present in the memory will be well detected by the anti-virus product.

The objective of this test is to validate the operation of the anti-virus product, regarding the read-write memory.

We use a server process program authorizing the exploitation of a buffer overflow (Frej & Ogorkiewicz, 2004). This type of vulnerability is used as a worm propagation vector. We inject *relocatable* viral code from a client program.

The following figure points out the principle of the exploitation of a buffer overflow. The call, within a function of the vulnerable program, with the function `strcpy()`, results in the save of the registers `eip2` and `ebp`.



The `strcpy()` function's prologue is as follows:

```
push ebp
mov ebp, esp
sub esp, BUF_LEN
push arg                ; input buffer address
push buf                ; memory buffer address
call f                  ; i.e. push eip1
```

The `f()` function fills the buffer from the bottom up with the chain pointed by `arg`, until meeting a null character at the end of the string. The `f()` function ends in a `ret` instruction which pops (depilate) `eip1` and resumes the course of the `strcpy()` function execution.

The end of the `strcpy()` function code is as follows:

```
add esp, 8              ; free buf and arg
mov esp, ebp            ;
pop ebp                 ;
ret                     ; pop buf instead eip2
```

The `eip` register points now towards the piece of relocatable code. Technically, it is difficult to predict the address of the buffer in memory. In our case, it is not a problem, insofar as the target service can communicate this address to us.

The anti-virus software must detect the presence of a data-processing infection in the memory. This test makes it possible to check whether a data-processing infection has occurred in the memory will be well detected by the anti-virus software.

TST_FMT (Management of the various file formats)

This test enables us to check if the anti-virus product suitably manages the various formats of file and compression, in particular when these formats are corrupted intentionally.

Some anti-virus products do not correctly analyze certain file formats. They are classified *unknown* instead of being decoded and being analyzed. An attacker can thus employ illicit files in order to pass on a virus.

We use GNU Win32 software to archive, compress or pack a virus recognized by the anti-virus product (ARC, ARJ, BASE64, BZIP2, GZIP, LHA, LZO, ms-compress, SHAR, TAR, UPX, UUE, ZIP file formats or extensions).

We carry out the analysis of the directory containing the packed, archived and/or compressed files. All the formats or encoding must be supported.

Case of UPX format: UPX (Ultimate Packer for eXecutables, 2006) format is used to compress a program and uncompress it before its execution. Some anti-virus products cannot recognize as viral such custom UPX files.

A remote attacker can thus create a virus, compress it in UPX and then apply a treatment to it so that it is still recognized by the system as being a UPX file, but not anymore by the anti-virus product (it is an evasion attack).

The antivirus software can consider the format as being unknown or the file as being corrupted and not declare the presence of a virus to the user. Consequently, the now confident user may decide to run it.

We have adapted the executable UPXRedir in order to be able to transform a UPX file into a file recognized as UPX by the system, but not by the anti-virus product (when this one does not manage this format in a correct way).

Thus the test consists of compressing a viral executable file with UPX format, ensuring that the anti-virus product detects the virus in this format, and then corrupting the executable file before again subjecting it to the analysis.

Case of the Zip format: By using a custom Zip file (Vigil@nce, 2006g, 2006h), an attacker can circumvent some anti-virus software.

The Zip files are compressed files containing one or more documents and/or directories. A header precedes each one of these documents. In the same way, a summary of all the files appears at the end of the archive. An attacker can modify the file sizes appearing in the header and the summary. Thus, an additional document can be hidden in the file.

WinZip and the Windows decompression utility extract the hidden file. However, some anti-virus products do not detect it. Any attacker can thus create a file containing a virus that will not be detected before reaching the user's workstation.

We have developed a tool that corrupts an archived file with the Zip format, while zeroing the *Uncompressed Size* fields of the *Local File Header* structures and the *Central Directory* of the Zip file.

Some anti-virus products do not properly inspect a file using the illicit format, and are thus vulnerable to this evasion attack.

Other file formats:

Zip and UPX formats are not the only formats that raise these problems. The following formats are also subject to evasion attack: RFC822 (Vigil@nce, 2006a), MIME (Vigil@nce, 2006b), LHA (Vigil@nce, 2006e), URI (Vigil@nce, 2006i), ANSI ESC (Vigil@nce, 2006j), RAR (Vigil@nce, 2006k). As a next step, we envisage building necessary tools to make sure that these formats are well managed by the anti-virus product.

TST_DOC (Filtering of the document viruses)

This test checks that an anti-virus product suitably detects the embedded viral code of an Office or HTML document.

Users often reactivate the execution of macros, scripts or the applet in an Office application or a navigator, to make the default configuration of the software more flexible. The anti-virus software must be able to notify the presence of viral code embedded in a document.

We have adapted the conformity tests suggested by eSafe to validate its script-filtering engine (eSafe, 2006).

TST_MAIL (Analysis of incoming and outgoing email)

This test validates functions analyzing incoming/outgoing emails. Anti-virus product shall monitor processes attempting to send or receive emails and may block traffic upon detection of a virus.

The scenario of the test is simple: send and receive an email containing an attached viral file.

According to the protection profile description of anti-virus actions (CCEVS, 2006) and the state of art in the field of anti-virus protection, we expect the anti-virus product to detect and disinfect or remove the attachment.

Initially, the procedure described above can be carried out with the EICAR test file (EICAR, 2006). This file is in fact a small 16-bit program which runs function 9 (print string to standard output) of the 21h interrupt (DOS services. ah=9 and ds:dx is a pointer to string to write, string terminated by a $ character). This program's binary code has the property of comprising only printable bytes. For information, EICAR test file is disassembled below.

```
seg000:0000 ; Base Address: 0000h Range: 0000h - 0046h Loaded length: 0046h
seg000:0000 ; Segment type: Pure code
seg000:0000 seg000          segment byte public 'CODE' use16
seg000:0000                 assume cs:seg000
seg000:0000                 assume es:nothing, ss:nothing, ds:nothing,
                                   fs:nothing, gs:nothing
seg000:0000                 pop     ax
seg000:0001                 xor     ax, 214Fh
seg000:0004                 push    ax
seg000:0005                 and     ax, 4140h
seg000:0008                 push    ax
seg000:0009                 pop     bx
seg000:000A                 xor     al, 5Ch
seg000:000C                 push    ax
seg000:000D                 pop     dx ; dx=011ch
seg000:000E                 pop     ax
seg000:000F                 xor     ax, 2834h
seg000:0012                 push    ax
seg000:0013                 pop     si ; si=ax=097bh
seg000:0014                 sub     [bx], si
seg000:0016                 inc     bx
seg000:0017                 inc     bx ; bx=0142h
seg000:0018                 sub     [bx], si
seg000:001A                 jge     near ptr word_40 ; ip=011ah, jge 0140h
seg000:001C aEicarStandardA db 'EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$'
seg000:0040 word_40         dw 2B48h ; → cd21h (int 21h, function 9)
seg000:0042                 dw 2A48h ; → cd20h (int 20)
seg000:0044                 db  0Dh
seg000:0045                 db  0Ah
seg000:0045 seg000          ends
seg000:0045                 end
```

The EICAR test file is a test file allowing vendors to calibrate their systems. It should be noted that a user without particular technical knowledge could carry out this test very easily.

In the second time, the procedure described above can be repeated with other viruses in attachment (highly compressed virus file, intentionally corrupted archive formatted virus files, viral code embedded in a document, etc.).

TST_VXDAT (Viral base analysis)

The objective of this test is to validate the completeness of the viral signature base of the anti-virus product.

We have a virus base (about 30000 uniquely identified viruses, at the time of analysis).

We set up the scanner in order to target the analysis on the directory containing the viral base.

In order to facilitate the maintenance of the viral base, we have developed a real-time monitor, which blocks the execution of the viral code (in the case of accidental access to a viral program) and alerts the administrator of the viral base.

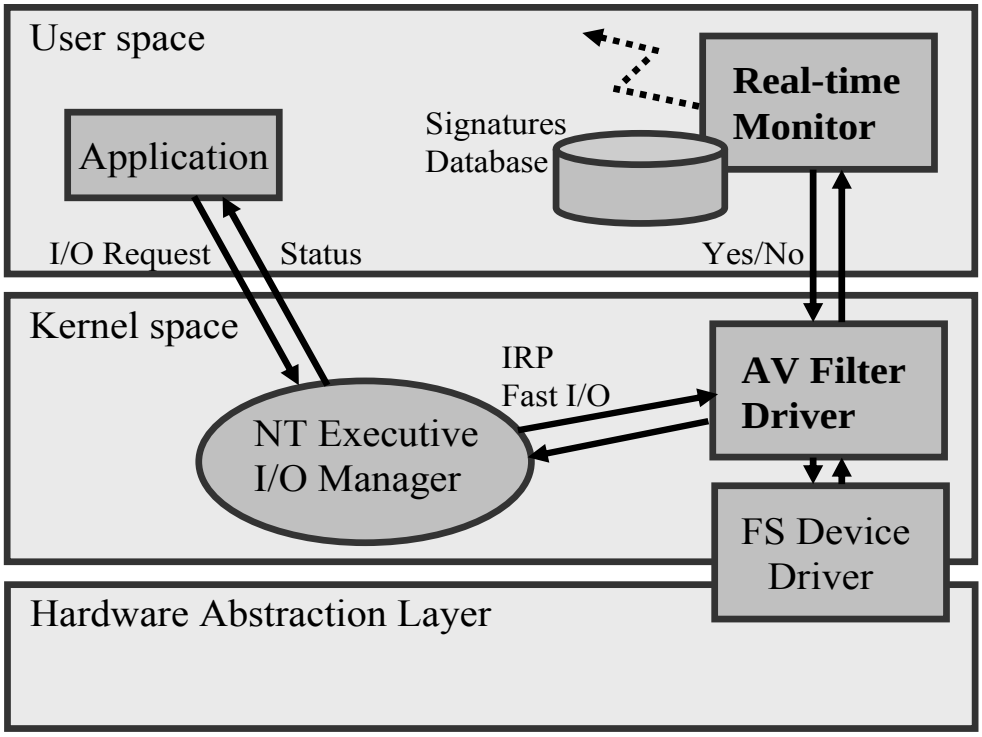
This real-time monitor embeds three components:

- a client program, which recovers the I/O carried out from user space towards the file systems and carries out their analysis while applying:

  - the Aho/Corasick pattern matching algorithm (Aho & Corasick, 1975; ClamAV, 2006) taking into consideration viral base signatures;
  - the FDA/NFA pattern matching algorithm taking into consideration a list of rational PCRE expressions (Hazel, 2003);

- a kernel mode driver, which makes it possible to intercept the IRP and the fast I/O, communicated by the I/O manager towards the file systems drivers (FAT, NTFS, RDR).

Once a virus is detected, the kernel driver blocks the access to the file system and the user is notified by the client program.



The client program proceeds with the loading of the signature base, with the initialization of the motif search structures, with the opening of a communication channel towards the driver and with the creation of threads. Each thread connected to the driver recovers data from it and carries out the analysis (by pattern matching). Upon virus detection, the thread raises an alarm.

Then the client program sends a message to the driver that results in imposing a blocking of the I/O request by the driver.

The driver is registered at the NT executive, and then creates a port communications server on which it can receive connection requests from the user mode. It includes in the port a security descriptor limiting access to the administrator. It can then intercept all the packets (IRP, Fast I/O) coming from the I/O manager towards the file system drivers (FAT, NTFS, RDR).

When an I/O request towards the file system is performed in user space, the driver retrieves information on the file, reads the beginning of this file, passes the contents into user space and sends an analysis request. If the file is suspect, then the I/O request is blocked.

If the file is not suspect and the I/O request corresponds to a file opening in write mode. Then the file will be analyzed again. The context is recovered. If a new analysis is required, the driver reads again the beginning of the file, passes the contents into the user space and sends a new analysis request. If the file is suspect, then the writing request is blocked.

This real-time monitor locks the files. A file opened by a process will thus not be able (before being closed) to be read and executed by another process (Das, Miretskiy, Wright & Zadok, 2004).

This program carries out on-demand analysis of whole or part of the viral base, with regard to the signature base.

We did neither explore the possibilities offered by the virus collection management tools, nor did we explore the possibilities offered by the use of an emulator, in order to check automatically that virus executables of a collection effectively run and correspond with their public description.

For example, the command line tool TTAnalyze (Bayer, 2005) seems well suited for analysing the interactions of a virus executable with the Windows system. The virus executable is run under a QEMU based emulated environment (Bellard, 2005). The dynamic analysis (on a virtual processor) analyzes self-modifying viruses, including packed and obfuscated executables. The QEMU emulator engine monitors the execution of the program and automatically gathers status information and interactions with the host system.

TST_UNICODE (Unicode Support)

This test checks that the anti-virus product implements the Unicode format suitably. The incorrect implementation of this format is at the origin of certain evasion attacks.

The objective of this test is to check if the anti-virus product detects viruses in long NT file system paths. Few anti-virus products may not be able to detect a virus that has an access path that is too long (Vigil@nce, 2006c).

The maximum size of the pathname is limited to 260 bytes for the directories and files created using the standard functions of API Win32. This maximum size is extended to 32767 bytes for directories and files created using the Unicode functions of API Win32.

Some anti-virus products are unable to detect viruses located in the directories whose path has a size greater than 260 bytes. Thus a virus may use this limitation to put a file which cannot be erased, and which may contaminate the machine each time it boots up.

In a general way, the anti-virus products that do not use the API Win32 Unicode functions are not able to detect viruses located in tree structures of more than 260 bytes.

We have developed an executable file which confines the viral code to the deepest part of a tree structure whose size is customizable. This executable file carries out the disinfection of the file system.

Each time the viral code is hidden too deeply in the tree structure of the file system, some anti-virus products are no longer able to detect its presence.

TST_AUDIT (Audit files)

The following events shall be recorded:

- start-up/shutdown of the audit security function;
- selection of an action by the user;
- action taken in response to the virus detection.

The audit security function shall record (within each audit record) at least the following information:

- date and time of the event;
- event type;
- subject identity (for audit events resulting from actions of identified users);
- outcome (success or failure) of the event;
- action selected by a user or action taken in response to the detection of a virus ;
- description of the detected virus, file or process identifier where the virus was detected.

The audit security function should protect the stored audit records from unauthorized modification/deletion via the audit security function interface (the environment (OS) is responsible for preventing unauthorised modification/deletion of the audit file via OS administration interfaces).

The audit security function shall provide the following options to prevent audit loss (if the audit trail is full):

- ignore auditable events;
- prevent auditable events, except those taken by the authorised user with special rights;
- overwrite the oldest saved audit records.

We can for example build a test to check the audit files rotation function of the anti-virus software, when this one imposes a limited size on the logs files. The scenario of the test is simple: fill the log files, and then check automatic rotations.

TST_INTEG (Integrity of the viral signature base)

This test aims at examining the robustness of the signature file. The signature file of the product is often written in a proprietary format. It should not be possible to modify the contents of the file.

We check the resistance of the product to restarting when one deteriorates the integrity of the viral signature base. The procedure is simple: we corrupt or affect the integrity of the viral

signature base. The objective is to check the mechanisms that guarantee the viral signature base integrity and to check the behaviour of the software, in terms of failure recovery.

We perform several actions:

- modification of the attribute: last modification date. Writing in one of the files of the viral signatures base;
- deletion of one of the signature base files;
- deletion of one of the signature base file entries;
- substitution of an old base to the new one;
- if several versions of the signature database are present in the installation directory of the product, remove and/or modify the most recent version.

In all cases, the refusal to load the base must be accompanied by an alarm or a notification. If a sound (but less recent) version is loaded, the anti-virus product must clearly announce the fact that it did not manage to reload the viral signature base or that it is running in degraded mode.

These tests thus validate the integrity auto-tests of each file of the viral signature base, the system of on-error recovery and the alarm system of the anti-virus product.

TST_UPDATE (Product updating)

An anti-virus product can be updated manually or by using the automatic update function. For the latter, the update is done on demand or in a scheduled way, at a frequency chosen by the user.

We use a network analyzer (*ethereal*) to trace how the automatic updates are carried out. The analysis of the Ethernet frames provides a better understanding of the update process: client request, server response listing the definition files available, their signatures and the name of a server which can provide them, finally downloading the definition files, when necessary.

We are waiting for the use of a protocol that authenticates the client, the update server, with possibly mutual authentication.

An analysis of the symbols (strings) of the update executable file may confirm the fact that the name of the server is hard-coded in the program.

We expect (after downloading of the update files of the signature base and possibly of the anti-virus product analysis engines) an integrity check on the downloaded files.

It may be necessary to restart the operating system or the anti-virus product. The (graphical) Man to Machine Interface must announce the modifications made into the viral signature base. The events related to the update of the product must be logged.

Conclusion and Future Works

We have proposed in this paper an approach and tools in order to help a single user or a company to be able to choose an anti-virus product off the shelf with a certain degree of confidence.

Both use of a protection profile (validated by the methodological approach of the Common Criteria) and the specification of security requirements validation tests seems to be a valuable basis to measure the confidence to grant an anti-virus product.

The main limitation induced by this approach is the objective interpretation of the tests. Even if the tests of our platform are automated, the results are sometimes firstly designed to enable the deepest investigation of the product internals. The verdict is not a binary *pass/do not pass*, but rather a verbose comment of the security analysts, as regards to the way the internal characteristics of an anti-virus product meet the state-of-the-art in the field of host based intrusion detection systems and the security requirements of its product category.

Our approach may obviously be optimized. The test set must evolve on a regular basis, in order to take into account the technical developments of the two protagonists involved. A constant technological watch must thus be conducted and especially looks for:

- evolutions of the viral threat, at the technical and algorithmic level;
- new techniques and anti-virus software architectures;
- vulnerabilities related to the design and the implementation of the anti-virus product and of its environment.

The protection profile (used as a methodological basis in this document) must also evolve to reinforce the level of security requirements and thus to increase the corresponding degree of confidence. A relevant test set will have to enable its implementation.

Appendices

Table 1 : Functional requirements

Functional requirement	Description
FAV_ACT_EXP.1	**Anti-virus actions**
FAV_ACT_EXP.1.1	Upon detection of a memory-based virus, the anti-virus security function shall prevent the virus from further execution.
FAV_ACT_EXP.1.2	Upon detection of a file-based virus, the anti-virus security function shall perform the actions specified by the administrator. Actions are administratively configurable on a per-workstation basis and consist of: clean the virus from the file, quarantine the file, and delete the file.
FAV_ACT_EXP.1.3	The anti-virus security function shall actively monitor processes attempting to access a remote system using TCP or UDP remote port 25 (SMTP) and block traffic from unauthorized processes.
FAV_ALR_EXP.1	**Anti-virus alerts**
FAV_ALR_EXP.1.1	An alert shall be displayed on the screen of the workstation on which the virus is detected and on the screen of the administrator (upon receipt of an audit event). The alerts shall continue to be displayed until they are acknowledged by the user or by the administrator.
FAV_ALR_EXP.1.2	
FAV_ALR_EXP.1.3	
FAV_ALR_EXP.1.4	
FAV_SCN_EXP.1	**Anti-virus scanning**
FAV_SCN_EXP.1.1	The anti-virus security function shall perform real-time scans for memory-based viruses and real-time, scheduled and on-demand scans for file-based viruses. The administrator shall perform scheduled scans at the time and frequency configured by the administrator. The workstation user shall perform manually invoked scans.
FAV_SCN_EXP.1.2	
FAV_SCN_EXP.1.3	
FAV_SCN_EXP.1.4	
FAU	**Security Audit**
FAU_GEN.1-NIAP-0347	The anti-virus security function shall be able to generate an audit record with suitable information, to provide the user and administrator with the capability to read all audit information, to perform and sorting of audit data. The stored audit records shall be protected from unauthorised deletion, modification or loss (if the audit trail is full).
FAU_SAR.1	
FAU_SAR.2	
FAU_SAR.3	
FAU_STG.1-NIAP-0429	

FAU_STG.NIAP-0414-NIAP-0429	
FCS	**Cryptographic support**
FCS_COP1	The anti-virus security function shall perform calculate a message digest to verify the integrity of the signature files in accordance with a NIST FIPS 140-2 approved cryptographic algorithm.
FMT	**Security management**
FMT_MOF.1	Management of security functions behaviour and data, specification of management functions and security roles.
FMT_MTD.1	
FMT_SMF.1	
FMT_SMR.1	
	Protection of the security functions
FPT_SEP_EXP.1	Partial security functions domain separation.

Table 2 : Environment requirements

ENVIRONMENT REQUIREMENT	**DESCRIPTION**
FAU	**SECURITY AUDIT**
FAU_STG.1-NIAP-0429	Protected Audit Trail Storage
FDP	**USER DATA PROTECTION**
FDP_RIP.1	Subset Residual Information Protection
FIA	**IDENTIFICATION AND AUTHENTICATION**
FIA_AFL.1	Authentication Failure Handling
FIA_SOS.1	Verification of Secrets
FIA_UAU.2	User Authentication Before any Action
FIA_UAU.6	Re-Authenticating
FIA_UID.2	User Identification Before any Action
FPT	**PROTECTION OF THE SECURITY FUNCTION**
FPT_ITT.1	Basic Internal security function Data Transfer Protection
FPT_RVM.1	Non-Bypassability of the security policy
FPT_SEP.1	Security function Domain Separation

FPT_STM.1	Reliable Time Stamps
FTA_SSL.1	Security function-Initiated Session Locking
FTA_TAB.1	Default anti-virus product Access Banners

Table 3 : Quality requirements

QUALITY REQUIREMENT	**DESCRIPTION**
ACM	**CONFIGURATION MANAGEMENT**
ACM_CAP.2	Configuration items
ADO	**DELIVERY AND OPERATION**
ADO_DEL.1	Delivery procedures
ADO_IGS.1	Installation, generation, and start-up procedures
ADV	**DEVELOPMENT**
ADV_FSP.1	Informal Functional specification
ADV_HLD.1	Descriptive high-level design
ADV_RCR.1	Informal correspondence demonstration
AGD	**GUIDANCE DOCUMENTS**
AGD_ADM.1	Administrator guidance
AGD_USR.1	User guidance
ALC	**LIFE CYCLE SUPPORT**
ALC_FLR.2	Flaw Reporting Procedures
ATE	**TESTS**
ATE_COV.1	Evidence of coverage
ATE_FUN.1	Functional testing
ATE_IND.2	Independent testing - sample
AVA	**VULNERABILITY ASSESSMENT**
AVA_MSU.1	Examination of guidance
AVA_SOF.1	Strength of anti-virus product security function evaluation
AVA_VLA.1	Developer Vulnerability analysis

References

Aho, A. V., & Corasick, M. J. (1975). Efficient string matching : an aid to bibliographic search. In Communication of the ACM, vol. 18, no. 6, pp. 333-340.

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S., & Yang, K. (2001, 19-23 August 2001). On the (im)possibility of obfuscating programs. In Advances in Cryptology - CRYPTO `01, vol. 2139 of Lecture Notes in Computer Science, pp. 1-18, Santa Barbara, CA.

Bayer, U. (2005). TTAnalyze: A tool for analyzing malware. Master's Thesis, Technical University of Vienna.

Bellard, F. (2005). QEMU, a Fast and Portable Dynamic Translator. In Proceedings of USENIX 2005 Annual Technical Conference, pp. 41-46.

Brubacher, D., & Hunt, G. (1999) Detours: Binary Interception of Win32 Functions. In Proceedings of the 3rd USENIX Windows NT Symposium, pp. 135-143, Seattle, WA.

CCEVS (2005). US Government Protection Profile Anti-Virus Applications for Workstations in Basic Robustness Environments. Version 1.0, 6 January 2005. Retrieved from : http://niap.nist.gov/cc-scheme/pp/PP_VID10053-PP.html

Chavez, P., Mukkamala, S., Sung, A. H., & Xu, J. (2004). Static Analyzer of Vicious Executables (SAVE). In Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04), pp 326-334.

Christodorescu, M., & Jha, S. (2003), Static Analysis of Executables to Detect Malicious Patterns. In Proceedings of the 12th Usenix Security Symposium, pp. 169-186.

ClamAV (2006). Retrieved from: http://www.clamav.net/

Cloakware, 2006. Retrieved from: http://www.cloakware.com/

Cogswell, B., Russinovich, M. (2006), Sysinternals. Retrieved from: http://www.sysinternals.com/

Cohen, F. (1986). Computer Viruses. Doctoral dissertation, University of Southern California, California, United States, 1986.

Common Vulnerabilities and Exposures (2006). Retrievd from: http://www.cve.mitre.org/

Cunningham, R. K., Khazan, R. I., Lewandowski, S. M., & Rabek, J.C. (October 2003). Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code. In Proceedings of the 2003 ACM Workshop on Rapid Malcode (WORM), Washington, DC, USA, pp. 76 – 82.

Dagon, D., Kolesnikov, O., & Lee, W. (2005) Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Georgia Institute of Technology, Technical Report.

Das, A., Miretskiy, Y., Wright, C. P., & Zadok, E. (2004) Avfs: An On-Access Anti-Virus File System. In Proceedings of the 13th USENIX Security Symposium.

Detoisien, E., & Dotan, E. (2003). Cheval de Troie furtif sous Windows : mécanismes d'injection de code. MISC Magazine n°10.

European Institute for Computer Anti-Virus Research (EICAR), 2006. Retrieved from: http://www.eicar.org/

eSafe (2006). eSafe test page. Retrieved from: http://www.esafe.com/home/csrt/eSafe_Demo/TestPage.asp

Eskin, E., Schultz, M. G., Stolfo, S. J., & Zadok, E. (2001). Data Mining Methods for Detection of New Malicious Executables. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, publisher: IEEE Computer Society, Washington, DC, USA.

Filiol, E., (2005). Strong Cryptography Armoured Computer Viruses Forbidding Code Analysis. In Proceedings of the 14th EICAR Conference, pp. 216—227.

Filiol, E., (2006). Malware Scanning Schemes Secure Against Black-box Analysis. In Proceedings of the 15th EICAR Conference. To appear.

Frej, P., & Ogorkiewicz, M. (2004). Analysis of Buffer Overflow Attacks. Retrieved from: http://www.windowsecurity.com/

GriYo (2006). EPO : Entry-Point Obscuring. Retrieved from: http://vx.netlux.org/lib/vgy01.html

Hazel, P. (2003). Perl-Compatible Regular Expressions. Retrieved from: http://www.pcre.org/

IFSKit (2006). Installable File System Kit. Retrieved from: http://www.microsoft.com/whdc/devtools/ifskit/

International Computer Security Association Labs (2006). Retrieved from: https://www.icsalabs.com/

Josse, S. (2005), Techniques d'obfuscation de code : chiffrer du clair avec du clair. MISC Magazine n°20, pp 32-42.

Low Level Virtual Machine (2006). Retrieved from: http://llvm.cs.uiuc.edu/

SandMark, 2006. Retrieved from: http://www.cs.arizona.edu/sandmark/

Security Focus Bugtraq (2006). Retrieved from: http://www.securityfocus.com/bid/

Szor P. (2005), Advanced Code Evolution Techniques and Computer Virus Generator Kits. Addison Wesley (Ed.).

Ultimate Packer for eXecutables (2006). Retrieved from: http://upx.sourceforge.net/

Vigil@nce (2006a). Outlook accepts messages whose format does not respect the RFC 822 (Standard for the format of ARPA Internet text messages). BUGTRAQ-5259, CVE-2002-0637. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006b). Incorrect analysis of MIME messages. BUGTRAQ-9650, CVE-2004-2088. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006c). Incorrect Unicode support. BUGTRAQ-10164. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006d). Denial of service by compressed files. BUGTRAQ-10537, BUGTRAQ-9393. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006e). Incorrect analysis of LHA files. BUGTRAQ-10243, CVE-2004-0234, CVE-2004-0235. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006f). Incorrect analysis of MS-DOS drives reserved names. BUGTRAQ-11236, BUGTRAQ-11328, CVE-2004-0552, CVE-2004-0920. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006g). Incorrect analysis of ZIP files when they are protected by a password or have several levels of overlap. BUGTRAQ-11600, BUGTRAQ-11669, BUGTRAQ-11732, CVE-2004-2220, CVE-2004-2442. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006h). No disinfection of ZIP file. BUGTRAQ-11448, CVE-2004-0932-0937, CVE-2004-1096. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006i). Incorrect analysis of the data integrated into a URI. BUGTRAQ-12269, CVE-2005-0218. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006j). Incorrect management of the files containing ANSI escape characters (these sequences can disturb display during the consultation of the audit files by the administrator). BUGTRAQ-12793. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006k). Incorrect analysis of RAR files. BUGTRAQ-13416, CVE-2005-1346. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006l). Skirting by using magic bytes (files with formats HTML, BAT, EML). BUGTRAQ-15189, CVE-2005-3370-3382. Retrieved from http://vigilance.aql.fr/

Vigil@nce (2006m). Incorrect analysis of the mbox, TNEF, CAB, FSG and OLE files. BUGTRAQ-15316, BUGTRAQ-15317, BUGTRAQ-15318, CVE-2005-3239, CVE-2005-3303, CVE-2005-3500, CVE-2005-3501. Retrieved from http://vigilance.aql.fr/

Virus Bulletin (2006). Retrieved from: http://www.virusbtn.com/

Vmware (2006). Retrieved from: http://www.vmware.com/

West Coast Labs (2006). Retrieved from: http://www.westcoastlabs.org/

WinDDK (2006). Windows NT Driver Devel Kit. Retrieved from: http://www.microsoft.com/whdc/driver/WDK/

Winpooch (2006). Retrieved from: http://winpooch.sourceforge.net/

Z0mbie (2001a). About Permutation (RPME). Retrieved from: http://vx.netlux.org/lib/

Z0mbie (2001b). Automated Reverse Engineering : Mistfall Engine. Retrieved from: http://vx.netlux.org/lib/

Malware Pattern Scanning Schemes Secure Against Black-box Analysis

Eric Filiol
Army Signals Academy - Virology and Cryptology Laboratory

About Author(s)

Eric Filiol is Head Scientist Officer of the Virology and Cryptology Laboratory. Contact Details: c/o Ecole Supérieure et d'Application des Transmissions, Laboratoire de virologie et de cryptologie , B.P. 18, 35998 Rennes, France, phone +33-2-99843609, fax +33-2-99843609, e-mail: efiliol@esat.terre.defense.gouv.fr

Keywords

Malware, malware pattern, antivirus, scanner, malware signature, black-box analysis, pattern extraction, malware detection.

Malware Pattern Scanning Schemes Secure Against Black-box Analysis

Abstract

As general rule, copycats produce most of malware variants from an original malware strain. For this purpose, they widely perform black-box analyses of commercial scanners aiming at extracting malware detection patterns. In this paper, we first study the malware detection pattern extraction problem from a complexity point of view and provide the results of a wide-scale study of commercial scanners black-box analysis. These results clearly show that most of the tested commercial products fail to thwart black-box analysis. Such weaknesses therefore urge copycats to produce even more malware variants. Then we present a new model of malware detection pattern based on Boolean functions and identify some properties that a reliable detection pattern should have. Lastly, we describe a combinatorial, probabilistic malware pattern scanning scheme that, on the one hand, highly limits black-box analysis and on the other hand can only be bypassed in the case where there is collusion between a number of copycats. This scheme can incidentally provide some useful technical information to malware crime investigators, thus allowing faster copycats'identification.

Introduction

Malware detection in most commercial scanners still highly relies on string pattern search. In contrast to the claims usually put forward by most of antivirus software publishers, practical experience mostly based on technical analysis of the available products, shows that sequence-based detection techniques are widely used:

- either as direct detection techniques (first generation techniques like string scanning, wildcards techniques, mismatches techniques....). As an illustration, this kind of techniques is widely used by *On-demand scanners*;
- or they take place as a final, decisive step once other detection techniques have been first applied, both in On-demand scanners or *On-access scanners* (second generation scanning techniques, malware specific detection techniques, static decryptor detection, code emulation...) (Szor, 2005).

Because of this above-mentionned strong dependence on malware pattern detection techniques – scanning for strings – many commercial antivirus software can be bypassed once a given malware pattern has been extracted and identified through a simple black-box analysis. This problem has previously been initiated in (Christodorescu & Jha, 2004) even though very simple instances of this problem have been taken into consideration.

A great deal of malware samples that are currently spreading nowadays are produced by copycats from an original malware strain or from its already detected variants[1]. Let us take a significant

[1]Virus prevalence table regularly illustrates this fact. As an example, February tables (Fortinet, 2006) show that more than 60 % of most prevalent malware were variants of known codes such as *W32/Bagle*, *W32/Netsky* or *W32/Mytob*.

example: more than thirty variants of *W32/Bagle* have been produced from the original code. This proliferation of malware variants not only makes life far more difficult from the computer users' point of view and affects the general security of networks but also significantly increases the malware analysts' amount of work in AV companies. These companies who have too much work as it is, do not stop updating their malware database[2] and share their technical information with other AV companies. As a result, the general computer and network protection are far from being acceptable as pointed out by the British Department of Trade and Industry in its 2004 Information Security Breaches Survey (Stimms, Potter & Beard, 2004).

More surprisingly, AV companies never made any attempt nor effort to thwart black-box analysis and copycats' activities by implementing more sophisticated detection schemes than those actually used. Somehow, this lack of effective technical attempts may be misinterpreted as an encouragement for copycats to produce malware variants. In this respect, while it is undoubtedly not the case, AV companies are nonetheless partly responsible for malware proliferation. It is all the more surprising that official, reference technical recommendation for protection profile (US Government Protection Profile, 2005) have not considered resistance against black-box analysis as a Target of Evaluation (TOE).

In this paper, we study the general problem of malware pattern extraction problem. For that purpose, we have first defined a new model for pattern detection based on Boolean functions under their disjunctive normal form (DNF). Moreover, we have identified some properties that efficient malware detection patterns should at least share such as resistance against pattern extraction. Our model enables black-box analysis and malware detection pattern to be considered as a DNF learning problem. While the problem of determining whether DNF formulae are learnable or not, is still an open problem, we show that tested commercial scanners represent very weak instances of this problem. The extraction of detection patterns is actually very easy perform.

We then present a new malware pattern scanning scheme that offers a good resistance level against black-box analysis under the assumption that a significantly large number N of copycats do not collude. We give some constructions in which the parameter N can be chosen according to the desirable level of resistance. Lastly, we provide possible implementations of this scheme which enable computer crime investigators to trace copycats that have been involved in the production and spreading of malware variants.

The paper is organised as follows. In Section , we first present our mathematical model for malware detection pattern and then detail the properties that reliable detection patterns should inevitably exhibit. Section is devoted to the malware pattern detection problem and presents some mathematical measures of the existing products. Section 0.0.1 deals with the detection pattern scanning scheme secure against black-box extraction. Finally, Section 0.0.1 contains both a general conclusion an a summary of different open problems which have been identified in relation to the malware detection problem.

Disclaimer.- The purpose of this paper is not to unnecessarily criticize commercial antivirus scanners which generally are rather efficient at managing known malware. The aim of this article is to identify some weaknesses in the detection schemes that are actually implemented with respect to

[2]During some malware outbreak periods or intense malware activity, we have observed on test platforms in our laboratory, that most AV software may be updated at least twice or thrice a day.

the production of new, undetected variants from already detected ones. The goal is also to propose efficient solutions that may thwart copycats'activities.

Mathematical Model for Malware Detection Pattern

In this section, our purpose is to define what a *malware pattern* (or *signature*[3]) is. Our definition slightly differs from generally accepted ones. The purpose is to consider at the same time both defence (efficient detection) and attack (copycats' action). Another advantage of our general definition is that it enables to study which properties should exhibit reliable malware detection patterns.

Mathematical Definition of Malware Patterns

Let us consider a file $\mathscr{F}$ of size n. It describes a potentially infected file or a malware sample. $\mathscr{F}$ is then a sequence of bytes, in other words, a sequence of n symbols over $\Sigma = \mathbb{N}_{255} = \{0,1,\ldots,255\}$. Hence $\mathscr{F}$ is an element of Σ^n.

We describe $\mathscr{S}_{\mathscr{M}}$ a malware pattern of size s with respect to a given malware $\mathscr{M}$ as a finite sequence of Σ^s. Thus

$$\mathscr{S}_{\mathscr{M}} = \{b_1, b_2, \ldots, b_s\}.$$

The i-th byte of $\mathscr{F}$ (respectively of $\mathscr{S}_{\mathscr{M}}$) will be denoted $\mathscr{F}(i)$ (resp. $\mathscr{S}_{\mathscr{M}}(i)$).

We say that $\mathscr{F}$ is infected by malware $\mathscr{M}$ if they are s locations in $\mathscr{F}$ (byte indices), denoted $\{i_1, i_2, \ldots, i_s\}$ such that

$$\mathscr{F}(i_j) = b_{\sigma(j)} \qquad 1 \leq j \leq s,$$

where σ denotes a bijective permutation over the byte of $\mathscr{S}_{\mathscr{M}}$. The use of permutation σ enables to take into account potential modification of $\mathscr{M}$ made by any copycat. Indeed, different code obfuscation or polymorphism/metamorphism techniques (Filiol, 2005; Szor, 2005) can modify the structure (byte ordering or indexing) of $\mathscr{S}_{\mathscr{M}}$:

- by modifying $\mathscr{S}_{\mathscr{M}}$ byte indexing (e.g. dummy code insertion). Then, $\sigma = Id_{\mathbb{N}_s^*}$;
- by modifying $\mathscr{S}_{\mathscr{M}}$ byte ordering (e.g. by obfuscation). Then $\sigma \neq Id_{\mathbb{N}_s^*}$. However the execution flow re-orders bytes of $\mathscr{S}_{\mathscr{M}}$ during the execution of $\mathscr{F}$. Second generation detection techniques or heuristics aim at bypassing permutation σ while simple malware pattern scanning (first generation detection techniques) operates the detection itself in a final step.

Thus in our approach, we consider the indices in $\mathscr{F}$ where the bytes of $\mathscr{S}_{\mathscr{M}}$ are effectively located (up to a permutation) rather than the malware pattern bytes themselves. Our notation describes in a better way the action of any copycat who tries to produce an undetected malware variant

[3]The term *malware pattern* or more precisely *malware detection pattern* will be preferred to the term *malware signature* or *signature* for short. Indeed, the latter term is used in some other contexts like cryptography with a very specialized meaning.

without too much effort. In a context of black-box analysis, we denote $\mathscr{S}_{\mathscr{F},\mathscr{M}}$ the set of indices $\{i_1, i_2, \ldots, i_s\}$.

Let us now mathematically describe the action of a given malware detector $\mathscr{D}$. Let us first define the s binary variables X_j ($1 \leq j \leq s$) as follows:

$$X_j = \begin{cases} 1 & \text{if } \mathscr{F}(i_j) = b_{\sigma(j)} \\ 0 & \text{otherwise.} \end{cases}$$

This notation enables to clearly describe the modification of $\mathscr{S}_{\mathscr{M}}$ bytes by any copycat who tries to bypass a given detector $\mathscr{D}$. Thus X_j equals 0 means that the copycat has indeed modified $\mathscr{S}_{\mathscr{M}}(j)$.

Let us now consider a Boolean function $f_{\mathscr{M}} : \mathbb{F}_2^s \to \mathbb{F}_2$ where $\mathbb{F}_2$ is the binary finite field. We say that a given detector decides that $\mathscr{F}$ is infected by the malware $\mathscr{M}$ with respect to the *detection function* $f_{\mathscr{M}}$ and the malware pattern $\mathscr{S}_{\mathscr{M}}$ is and only if $f_{\mathscr{M}}(X_1, X_2, \ldots, X_s) = 1$. In other words:

$$f_{\mathscr{M}}(X_1, X_2, \ldots, X_s) = \begin{cases} 1 & \mathscr{F} \text{ is infected by } \mathscr{M} \\ 0 & \mathscr{F} \text{ is not infected by } \mathscr{M}. \end{cases}$$

We will represent detection functions by their disjunctive normal form (DNF) that is to say the logical disjunction of terms, each term being a conjunction of literals X_i. By construction, literals do not appear under negated form $\overline{X_i}$. Thus, detection functions are modelled by *monotone* DNF. We will see in Section that any scanning scheme can be described by a detection function $f_{\mathscr{M}}$.

Since copycats obviously show a great interest in the different ways of bypassing any given malware detection pattern, we will consider the *non-detection* function $\overline{f_{\mathscr{M}}}$ as the negation of the detection function $f_{\mathscr{M}}$ instead. In other words $\overline{f_{\mathscr{M}}} = 1 \oplus f_{\mathscr{M}}$. This function describes the way malware pattern bytes can be modified to bypass detection with respect to $f_{\mathscr{M}}$. In the rest of the paper, we will indifferently consider either the detection function or the non-detection function.

Remark.- The malware $\mathscr{M}$ is uniquely characterized by both $\mathscr{S}_{\mathscr{M}}$ and the detection function $f_{\mathscr{M}}$. This function allow to greatly reduce the false positive rate.

In order to have a compact description of how sequence-based detection actually works, let us propose the two following definitions.

Definition 1 *A malware* detection scheme *with respect to a given malware $\mathscr{M}$ is a pair $\{\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}}\}$. If we consider sequence-based detection the set $\mathscr{S}_{\mathscr{M}}$ will contain bytes whereas if function-based detection is considered instead, this set contains program functions.*

Definition 2 *A malware* bypassing scheme *with respect to a given malware $\mathscr{M}$ is a pair $\{\mathscr{S}_{\mathscr{M}}, \overline{f_{\mathscr{M}}}\}$. If we consider sequence-based detection the set $\mathscr{S}_{\mathscr{M}}$ will contain bytes whereas if function-based detection is considered instead, this set contains program functions.*

Properties of Malware Patterns

Many questions arise about the kind of properties that a reliable malware detection pattern must exhibit. What are the best parameters to consider? Besides the problem of choosing good patterns, having some mathematical measures enables existing commercial scanners to be evaluated. As a matter of fact, antivirus software evaluation is mostly a subjective process which greatly differs from a test to another. Here are some properties that we have identified. Section will be devoted to the accurate evaluation of these properties.

Malware pattern entropy and transinformation

The purpose is to determine the uncertainty that a detector black-box analyst must face when trying to extract a given malware pattern $\mathscr{S}_{\mathscr{M}}$ with respect to a given detector $\mathscr{D}$.

For that purpose, we use the entropy function as defined by C. E. Shannon (Shannon, 1948). Given a random variable X that takes a finite set of values with probabilities $p_1, p_2, \ldots, p_n$, the uncertainty or entropy of X is defined by:

$$H(X) = -\sum_{k=1}^{n} p_k \log_2(p_k).$$

In the present context, the variable X represents $\mathscr{S}_{\mathscr{F},\mathscr{M}}$ and any of its parameters. Indeed, the analyst has no information about any of the parameters of the detection pattern (its size s, the pattern bytes or equivalently their location in $\mathscr{F}$ and the detection function $f_{\mathscr{M}}$).

The difficulty to extract $\mathscr{S}_{\mathscr{F},\mathscr{M}}$ through a black-box analysis increases with $H(\mathscr{S}_{\mathscr{F},\mathscr{M}})$ and $H(\mathscr{S}_{\mathscr{F},\mathscr{M}}) = 0$ when no incertainty exists.

Since in our case computing $H(X)$ is very complex, we will use the concept of *mutual information* or *transinformation* instead. If we consider that $\mathscr{S}_{\mathscr{M}}$ and $f_{\mathscr{M}}$ are random variables (for the analyst) we then define:

$$I(\mathscr{S}_{\mathscr{F},\mathscr{M}}; f_{\mathscr{M}}, \mathscr{S}_{\mathscr{M}}) = H(\mathscr{S}_{\mathscr{F},\mathscr{M}}) - H(\mathscr{S}_{\mathscr{F},\mathscr{M}} | f_{\mathscr{M}}, \mathscr{S}_{\mathscr{M}})$$

as the amount of information that $f_{\mathscr{M}}$ and $\mathscr{S}_{\mathscr{M}}$ together reveal about $\mathscr{S}_{\mathscr{F},\mathscr{M}}$. In other words, the transinformation describes the consequence for the analyst to have a detector $\mathscr{D}$ at his disposal ($\mathscr{D}$ contains and leaks all the information about $f_{\mathscr{M}}$ and $\mathscr{S}_{\mathscr{M}}$).

By construction, it is obvious to define the same property when considering the non-detection function $\overline{f_{\mathscr{M}}}$. Then

$$I(\mathscr{S}_{\mathscr{F},\mathscr{M}}; f_{\mathscr{M}}, \mathscr{S}_{\mathscr{M}}) = I(\mathscr{S}_{\mathscr{F},\mathscr{M}}; \overline{f_{\mathscr{M}}}, \mathscr{S}_{\mathscr{M}}).$$

Malware pattern resistance

Once the analyst has succeeded in extracting the pattern $\mathscr{S}_{\mathscr{M}}$ of a given scheme $\{\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}}\}$, he tries to bypass it in order to produce a non-detected variant. We then define the detection scheme *resistance*, denoted $\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}})$ as the more or less difficulty that a copycat has to face in order to bypass it. This property obviously depends not only on the size of the set $\mathscr{S}_{\mathscr{M}}$, but also on the weight of the detection function $f_{\mathscr{M}}$ (denoted $\mathrm{wt}(f_{\mathscr{M}}) = \{X \in \mathbb{F}_2^s | f_{\mathscr{M}}(X) = 1\}$):

- the number of possible modifications that enable the copycat to effectively produce a non-detected variant increase with the size of $\mathscr{S}_{\mathscr{M}}$;
- the lower the weight of the detection function, the easier it is to bypass the search mode with respect to $\mathscr{S}_{\mathscr{M}}$.

Thus, according to these carateristics, we define the scheme resistance as follows:

$$\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}}) = \frac{\{X=(X_1,X_2,\ldots,X_s)|f_{\mathscr{M}}(X)=0\}}{s2^s} \quad (1)$$

$$= \frac{\{X=(X_1,X_2,\ldots,X_s)|\overline{f_{\mathscr{M}}}(X)=1\}}{s2^s}. \quad (2)$$

Consequently, we have:

$$0 \leq \mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}}) \leq 1,$$

The lower $\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}})$, the easier the detection scheme bypassing is. In the extreme, there is no record for the malware $\mathscr{M}$ in the viral pattern database if $\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}}) = 0$.

It it worth noticing that $\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}})$ relates to the number of possible byte modification configurations (since $X_i \in \{0,1\}$). A copycat may modify every byte involved in a given configuration (e.g. an entry of the detection function) and replace it with any of the 255 remaining values. Thus, the effective grand total of possible byte modifications tremendously increases. This number conversely relates to $\mathscr{R}(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}})$. More precisely, if we denote $X=(X_1,X_2,\ldots,X_s) \in \mathbb{F}_2^s$ and if $\mathrm{wt}(X)$ denote the Hamming weight of X (in other words the number of ones in the binary representation of X), then the grand total Δ of possible byte modifications that may be performed by a copycat is given by:

$$\Delta = \sum_{X in \mathbb{F}_2^s} \overline{f_{\mathscr{M}}}(X).(256^{s-\mathrm{wt}(X)}-1). \quad (3)$$

We will examine some particular cases in Section 0.0.1.

Malware pattern efficiency

The efficiency of a malware pattern relates to its capacity to uniquely detect and identify a given malware. Moreover, any detection pattern must be frameproof with respect to other malware and any other (non-infected) file. This last requirement is linked both with false alarm probability (better known as false positive probability) and to malware phylogeny issues as defined in (Goldberg et al., 1998; Karim Md et al., 2005).

Previous results (Kephart & Arnold, 1994) showed that determining false positive probability is not as easy as it seems. Typically, malware pattern size ranges between 12 and 36 (Kephart & Arnold, 1994). The probability for a s-byte sequence to be present in a text is $\frac{1}{256^s}$. Thus as soon as s is large enough, this probability tends towards 0 and theoretically no file could be falsely detected as infected unless it indeed contains the malware code. Unfortunately, theory greatly differs from practice. As pointed out in (Kephart & Arnold, 1994), there is 34% chance of finding a random 24-byte sequence twice in a corpus of approximately half a gigabyte whereas theory of probability gives a 0.85×10^{-49}.

This discrepancy comes from the fact that bytes in an executable file cannot be described by independent, identically distributed variables (of probability $p = \frac{1}{256}$. As an example, we have $P[b_0 =' M'] = 1$ and $P[b_1 =' Z'|b_0 =' M'] = 1$ in a Win32-executable. There exists a strong dependence between bytes in an executable code. Markov processes appear to be a better model when it comes to describe probabilistic behaviour of executable bytes.

As pointed out in (Kephart & Arnold, 1994), "*Cleverness, be it human or algorithmic, is an essential ingredient in choosing good computer viruses signatures*." Most of commercial scanners use relatively efficient malware patterns, up to a more or less limited false positive probability.

Remark.- The malware pattern efficiency primarily depends on the pattern size. This is the reason why some commercial scanners are more sensitive to false positive probability than others due to pattern sizes which results too small. Moreover, the detection function may have some effect on the false positive probability as well. This point is still an open problem.

The Detection Pattern Extraction Problem and the Mathematical Evaluation of Viral Pattern Scanners

Viral Pattern Extraction

AV scanners efficiency has recently been studied by Christodorescu and Jha (Christodorescu & Jha, 2004). However their malware pattern extraction algorithm exhibit limitions. In addition to minor algorithmic flaws[4], their extraction algorithm considers only a limited number of detection function classes. We are now presenting two extraction algorithms, the second one being able to manage any detection function classes.

Given a malware detector $\mathscr{D}$, we aim at extracting the detection pattern of a given malware $\mathscr{M}$, with respect to $\mathscr{D}$. The extraction process is performed through a black-box analysis. No reverse-engineering is required. According to our previous notation, we must retrieve the non-detection function $\overline{f_{\mathscr{M}}}$ and the pattern bytes indices. Since the file $\mathscr{F}$ is the malware sample $\mathscr{M}$ itsef, we denote $\mathscr{S}_{\mathscr{M},\mathscr{M}}$ instead of $\mathscr{S}_{\mathscr{F},\mathscr{M}}$.

We consider the non-detection function rather than the detection function for the following reasons:

- in order to evaluate resistance of malware scanners with respect to black-box analysis, the copycat's point of view yields more information;
- extracting the detection function would require to compute its negation in a second step. Unfortunately, computing the negation of a DNF formula has exponential worst case complexity[5]. Consequently, Algorithm E-2 will directly extract the non-detection function.

The Naive Approach: Algorithm E-1

A first algorithm has been considered to extract the most frequent detection pattern and the most common detection function. As compared with the algorithm given in (Christodorescu & Jha, 2004), it may seem, at first sight, less efficient and rather naive. However, practical experience

[4] In particular, we identified some pattern configurations that lead to infinite loop in the FINDLEFTMOST and FINDRIGHTMOST procedures.

[5] Computing the negation of a DNF formula is equivalent to turn a conjuntive normal form into its corresponding disjunctive normal form. This transformation has exponential worst-case complexity (Papadimitriou, 1995, Theorem 4.1).

actually showed it was not. On the contrary, it succeeded in systematically extracting detection patterns as well as its corresponding detection function whereas the Christodorescu and Jha's algorithm failed to do it in a few cases in which the detection pattern contained bytes scattered over the malware code.

Let us consider a malware code $\mathscr{M}$ of size n. Here follows our first extraction algorithm. The

Input: A malware sample $\mathscr{M} = \{m_1, m_2, \ldots, m_n\}$, a malware detector $\mathscr{D}$ and a malware name σ.
Output: The malware pattern $\mathscr{S}_{\mathscr{M},\mathscr{M}}$ (pattern indices).

$\mathscr{S}_{\mathscr{M},\mathscr{M}} \leftarrow \{\}$
for i = 1 to n do
    modify only byte m_i in $\mathscr{M}$
    If $\mathscr{D}(\mathscr{M}) \neq \sigma$ then
        $\mathscr{S}_{\mathscr{M},\mathscr{M}} \leftarrow \mathscr{S}_{\mathscr{M},\mathscr{M}} \cup \{m_i\}$
    end if
end for
return $\mathscr{S}_{\mathscr{M},\mathscr{M}}$

Table 1: Malware Detection Pattern Extraction Algorithm E-1

algorithm successively modifies each bytes of the malware sample $\mathscr{M}$ (the byte is xored with a constant value) and performs a detection scanning with $\mathscr{D}$. If the detector still detects $\mathscr{M}$ (with our notation $\mathscr{D}(\mathscr{M}) = \sigma$), the modified byte is not involved in the detection pattern. On the contrary, the byte index belongs to $\mathscr{S}_{\mathscr{M},\mathscr{M}}$.

The complexity of Algorithm 1 is linear in the malware size that is to say $\mathscr{O}(n)$. The main interest of Algorithm 1 lies in its capacity to manage malware pattern whatever their structure may be (sparse number of bytes, bytes randomly scattered over the code...). Unfortunately, its efficiency is limited to a single kind of detection function whose DNF formula is given by

$$f_{\mathscr{M}}(X_1, X_2, X_3, \ldots, X_s) = X_1 \wedge X_2 \wedge X_3 \wedge \ldots \wedge X_s.$$

This detection function corresponds to the most currently used detection technique: string scanning with no wildcards (first generation scanning techniques). We will use the AND function to denote this detection function. Contrary to what most commercial scanners publishers are generally claiming, the results show that this technique is still widely used. Appendix 0.0.1 presents detailed results about some variants of the *W32/Bagle* family.

Remark.- The non-detection function of the AND detection function is very easy to compute using simple Boolean calculus rules. We then have:

$$\overline{f_{\mathscr{M}}}(X_1, X_2, X_3, \ldots, X_s) = X_1 \vee X_2 \vee X_3 \vee \ldots \vee X_s.$$

Input: A malware sample $\mathscr{M} = \{m_1, m_2, \ldots, m_n\}$, a malware detector $\mathscr{D}$ and a malware name σ.
Output: The malware pattern $\mathscr{S}_{\mathscr{M},\mathscr{M}}$ (pattern indices) and the non-detection function $\overline{f_{\mathscr{M}}}$ DNF.

$\mathscr{S}_{\mathscr{M},\mathscr{M}} \leftarrow \{\}$; $\text{DNF}_{f_{\mathscr{M}}} \leftarrow \{\}$
$S \leftarrow 2^{\lfloor \log_2(n) \rfloor + 1}$; $S' \leftarrow 2^{\lfloor \log_2(n) \rfloor + 1}$
While $S > 0$ do
    $S \leftarrow \frac{S}{2}$; $q \leftarrow \frac{S'}{S}$
    For i = 0 to q -1 do
        binf $\leftarrow i \times S$; bsup $\leftarrow$ binf $+ S$
        modify bytes in $I =$ [binf, bsup[in $\mathscr{M}$
        If $\mathscr{D}(\mathscr{M}) \neq \sigma$ then
            $\mathscr{S}_{\mathscr{M},\mathscr{M}} \leftarrow \mathscr{S}_{\mathscr{M},\mathscr{M}} + I$
            $X = 1_I(X_1, X_2, \ldots, X_n)$
            $\text{DNF}_{f_{\mathscr{M}}} \leftarrow \text{DNF}_{f_{\mathscr{M}}} \cup X$
        End If
    End For
End While
$\mathscr{S}_{\mathscr{M},\mathscr{M}} \leftarrow \text{COMBINATORIALMINIMIZE}(\mathscr{S}_{\mathscr{M},\mathscr{M}})$
$\text{DNF}_{f_{\mathscr{M}}} \leftarrow \text{LOGICALMINIMIZE}(\text{DNF}_{f_{\mathscr{M}}}, \mathscr{S}_{\mathscr{M},\mathscr{M}})$

Table 2: Malware Detection Pattern Extraction Algorithm E-2

This non-detection function is the OR function. Modifying any of the pattern bytes enables the malware to remain undetected.

DNF Formulae Learning Approach: Algorithm E-2

In order to extract any malware detection pattern and whatever the detection function may be (or equivalently the non-detection function), we are now considering another, more powerful extraction algorithm based on learning techniques. Let us first recall some fundamental concepts in Learning theory. The interested reader will refer to (Kearns & Vazirani, 1994) for a detailed presentation on the subject.

We will focus on *Boolean concept learning* in which the learner's goal – in other words any black-box analysts in our context – is to infer how an unknown target function – the non-detection function in our case – classifies (as positive or negative) examples from a given domain. The *instance space* or *domain* $\mathscr{X}$ is the set of all possible objects (instances) to be classified – patterns bytes of a malware detection pattern. We will consider the Boolean hypercube $\{0,1\}^n$ as reference domain. This corresponds to the non-detection function inputs as defined in Section . A *concept*

is a Boolean function over domain $\mathscr{X}$. A *concept class* $\mathscr{C}$ is a collection of subsets of $\mathscr{X}$. In other words $\mathscr{X} \subseteq 2^{\mathscr{X}}$. In our case study, $\mathscr{C}$ describes the set of all possible non-detection function DNFs. Then each $x \in \mathscr{X}$ is classified according to membership of a *target concept* $f \in \mathscr{C}$ – the malware non-detection function $f_{\mathscr{M}}$. A point $x \in \mathscr{X}$ is a *positive* example of f if $f(x) = 1$ or a *negative* example of f otherwise. This case of learning method is denoted *Query Learning Model*.

In our context, the DNF formula to learn is the non-detection function $f_{\mathscr{M}}$ DNF. Each literal X_i describes a byte of the malware sample $\mathscr{M}$ where $i = 1, 2, \ldots, n$ ($n = |\mathscr{M}|$). A DNF is then the union of conjonctions of literals that may be either X_i or $\overline{X_i}$ (see in Appendix 0.0.1 the exact meaning of this notation).

It is worth noticing the following important point. The problem of learning general DNF formulae is a long-standing open problem even if efficient learning methods have been found for some particular DNF subclasses. However, the time and memory complexity of any learning algorithm obviously highly depends on the complexity of the underlying target concept. In the case of Boolean DNF formulae, it depends on the number of terms which ranges from 0 (the null function) to 2^n (the constant function $f(x) = 1$). The AND detection function that we have just considered in Section contains a single term. This explains why Algorithm E-1 is suitable for this very particular function and is better than any other learning algorithm whose complexity is likely to be very high.

Our DNF formulae learning extraction algorithm is presented in Table 2. For the sake of clarity, we provide here a non-recursive version. The notation $X = 1_I(X_1, X_2, \ldots, X_n)$ describes the construction of a logical term in the DNF as follows using the characteristic function with respect to the (support) interval I. Each literal X_i or its negated form appears according to the following rule:

$$X_i = \begin{cases} \overline{X_i} & \text{if } i \in I \\ X_i & \text{if } i \notin I \end{cases}$$

Algorithm 2 is organised in three parts:

1. the first part is an initial learning step. It consists in modifying sections of contiguous bytes of malware code $\mathscr{M}$. These modifications are made through a dichotomic approach in $\log_2(n)$ steps. Sections of bytes are of decreasing length (a power of two). As in the Christodorescu and Jha's algorithm (Christodorescu & Jha, 2004), the main purpose is to first identify the bytes which are involved in the detection pattern $\mathscr{S}_{\mathscr{M},\mathscr{M}}$. This step enables to find some logical terms of the non-detection DNF. This first step has complexity in $\mathscr{O}(2n)$ (WHILE loop);

2. the second part, denoted COMBINATORIALMINIMIZE, is a combinatorial minimisation step. Its aim is to eliminate the DNF terms which are redundant. The set $\mathscr{S}_{\mathscr{M},\mathscr{M}}$ which has been output by the previous step generally contains a few terms whose support interval (see previous notation) are included in some other support interval. To illustrate this point, logical terms $X_1X_2X_3X_4$ and X_1X_2 have support intervals $[1,4]$ and $[1,2]$ respectively. The second one is included in the first one. This implies that variables X_3 and X_4 are not involved in the detection pattern. We only keep the X_1X_2 term in the DNF. Thus, this combinatorial minimisation step is keeping the minimal elements in poset whose partial ordering is set inclusion. This step has a worst-case complexity in $\mathscr{O}(t\log(t) + tn)$ (a sort plus a comparison

of consecutive terms) where $t = |\mathscr{S}_{\mathscr{M},\mathscr{M}}|$ before entering the COMBINATORIALMINIMIZE procedure. The result is a set of size s;

3. the LOGICALMINIMIZE procedure whose purpose is two-fold:

   - to complete the DNF learning process by exhaustively trying any s-tuple of variables. With the previous notations, to any s-tuple of $\mathbb{F}_2^s$ corresponds a byte modification configuration in $\mathscr{S}_{\mathscr{M},\mathscr{M}}$. The $\mathscr{M}$ code is then modified according this configuration and then tested with respect to the detector $\mathscr{D}$. If the code is no longer detected, the logical term corrsponding to this configuration is added to the DNF formula;
   - then, a logical minimisation step is performed. Indeed, the output DNF contains terms which are logically redundant. Consequently, by applying Boolean calculus (Denis-Papin, Kaufmann & Faure, 1963; Michaels, 2000) and the Quine-McCluskey method to simplify Boolean expressions (McCluskey, 1956; Quine, 1952; Quine, 1955). The logical minimisation problem is NP-hard (Michaels, 2000, Chap. 5.8.3). It has a complexity in $\mathscr{O}(s2^s)$ (Wegener, 1987).

   Finally, the LOGICALMINIMIZE procedure has a complexity in $\mathscr{O}(2^s + s2^s)$ if $s = |\mathscr{S}_{\mathscr{M},\mathscr{M}}$ before entering this step.

As a conclusion, we have the following result.

Theorem 1 *The algorithm 2 given in Figure 2 succeeds in extracting the detection scheme of a malware $\mathscr{M}$ of size n whose detection pattern has size s with a time complexity in $\mathscr{O}(sn + s2^s)$.*

Proof.
When considering the previous elements, the general (worst-case) complexity is in $\mathscr{O}(2n + t\log(t) + tn + s^2 2^s)$. The terms $2n$ and t are negligible compared to 2^s and we have $t \equiv s$ as well (as confirmed by our experiments). Hence the result. □

Remarks.-

1. Algorithm 1 is a particular case of Algorithm 2 when the detection function is the AND function. Being the most frequent case, using Algorithm 1 is a better choice since we suppress the logic and combinatorial minimization steps.

2. Learning the monotone DNF formula of the detection function requires a far better complexity when using Angluin's algorithm (Angluin, 1988). Using membership and equivalence queries, this algorithm exactly learns an unknown m-term monotone DNF formula over the domain $\{0,1\}^n$ with only $\mathscr{O}(nm)$ membership queries overall. But the DNF negation step must then be applied to get the non-detection function. This step has exponential worst-case complexity.

3. Algorithm E-2 succeeds in finding the exact non-detection function DNF unlike most learning methods which only produce equivalent function DNF.

Examples of Detection Functions

To any string based detection method corresponds a detection function DNF whose literals are bytes of any malware undergoing a scanning process. In order to illustrate this point, let us consider some examples of pattern-based detection techniques that can be described by detection function DNFs which are different from the AND function.

Wildcard detection

Wildcard detection generally considers pattern in which variable bytes indices are involved. Let us consider the following detection pattern extract taken from (Szor, 2005, Section 11.1.2).

```
..... 07BB ??02 %3 33C9 ......
```

This detection pattern extract then is represented by the following detection function DNF (extract):

$$\begin{aligned} \ldots \quad & (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02) \wedge (X_{i+4} = 33) \wedge (X_{i+5} = C9) \\ \vee \quad & (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02) \wedge (X_{i+5} = 33) \wedge (X_{i+6} = C9) \\ \vee \quad & (X_i = 07) \wedge (X_{i+1} = BB) \wedge (X_{i+3} = 02) \wedge (X_{i+6} = 33) \wedge (X_{i+7} = C9) \\ \ldots & \end{aligned}$$

Mismatches techniques

This techniques was developped by IBM as part of its antivirus scanner research. Mismatches techniques allow N number of bytes in the detection pattern to represent any value, regardless of their byte location indices in the pattern. Let us consider the following detection pattern `01 02 03 04` with mismatch value of 2 (this example has been extracted from (Szor, 2005, Section 11.1.3). Then the corresponding detection function DNF (extract) is given by:

$$\begin{aligned} \ldots \quad & (X_i = 01) \wedge (X_{i+1} = 02) \vee (X_i = 01) \wedge (X_{i+2} = 03) \\ \vee \quad & (X_{i+1} = 02) \wedge (X_{i+2} = 03) \vee (X_i = 01) \wedge (X_{i+3} = 04) \\ \vee \quad & (X_{i+1} = 02) \wedge (X_{i+3} = 04) \vee (X_{i+2} = 03) \wedge (X_{i+3} = 04) \end{aligned}$$

In this case, for a pattern of size s with mismatch value μ ($\mu < s$), it is obvious that the DNF formula contains $\binom{s}{\mu}$ terms, each of them having $s - \mu$ literals (it is called a $(s-\mu)$-DNF).

0.0.1 Nearly exact identification

This method is used to increase the accuracy of detection. Instead of considering only one detection pattern, we consider two such patterns (Szor, 2005, Section 11.2.3). In other words, we have to consider in this case to detection schemes $(\mathscr{S}^1_{\mathscr{M}}, f^1_{\mathscr{M}})$ and $(\mathscr{S}^2_{\mathscr{M}}, f^2_{\mathscr{M}})$. It is easy to prove that this case can be described by a unique scheme $(\mathscr{S}_{\mathscr{M}}, f_{\mathscr{M}})$ such that $\mathscr{S}_{\mathscr{M}} = \mathscr{S}^1_{\mathscr{M}} \cup \mathscr{S}^2_{\mathscr{M}}$ and $f_{\mathscr{M}} = f^1_{\mathscr{M}} \vee$

$f^2_{\mathscr{M}}$ or $f_{\mathscr{M}} = f^1_{\mathscr{M}} \wedge f^2_{\mathscr{M}}$ according to the way both schemes are combined (up to a minimization step).

This case also applies when two or more detection engines are involved. Our model enables to consider a single engine instead.

Other detection techniques

Other sequence-based techniques like *bookmarks techniques*, *smart scanning*, *skeleton detection*, *static decryptor techniques*, *sequence-based heuristics*..., among others (see (Szor, 2005, Chap. 11) can be similarly described. In these cases:

- the parameter *s* (the number of bytes involved) is very large (the detection pattern spans the whole code);
- the detection function may be generally far more complex (the number of logical terms is close to 2^s). However, it is not sure that the corresponding non-detection function is complex as well. This is an open problem (see section 0.0.1); detection pattern spans the whole code;
- particular combinatorial structures in the detection function DNF may be considered in addition to its weight.

As far as hashing or checksums techniques are considered, our approach can be easily generalized by considering bitwise approach instead of a bytewise approach. Any checksum value or hashing value can be described as a set of Boolean functions as demonstrated in (Filiol, 2002). The resulting detection function is then described as the logical intersection of these functions (Filiol, 2006).

Mathematical Measures of AV Scanners

Algorithms 1 and 2 enable to systematically extract detection pattern of the malware samples we used for testing (see Appendix 0.0.1). The tested commercial scanners have shown some very interesting but yet surprising results. As a general result, we can assert that the overall quality of detection pattern (including the detection and non-detection functions) tends to be weak not to say extremely weak in some cases – for the products we have tested. The main results are summarized hereafter.

Malware pattern transinformation

Whatever the commercial scanner we considered may be, the extraction of different malware pattern parameters (non-detection function $\overline{f_{\mathscr{M}}}$, pattern size *s* and pattern bytes indices set $\mathscr{S}_{\mathscr{M},\mathscr{M}}$) has been easily performed. In other words, as the scanner leaks all the information on the detection pattern, the transinformation value is consequently maximal. Thus we can write

$$I(\mathscr{S}_{\mathscr{M},\mathscr{M}}; \overline{f_{\mathscr{M}}}, \mathscr{S}_{\mathscr{M}}) = H(\mathscr{S}_{\mathscr{M},\mathscr{M}}).$$

This means that the only thing to do for a copycat who wants to get rid of the problem of uncertainty as detection pattern is concerned, is to simply get the detector to bypass. In this respect, all the commercial scanners that we have tested are rather weak.

Malware pattern resistance

The scheme resistance quantifies the copycat's effort to bypass a given detection scheme, once the latter has been successfully extracted. This property relates to the total number of possible byte modifications in order to delude a given detector $\mathscr{D}$. Equations 1 and 2, in Section show that both the parameter s and the detection function $f_{\mathscr{M}}$ (or equivalently the non-detection function $\overline{f_{\mathscr{M}}}$) have an impact on this number.

The precise evaluation of the scheme resistance require to enumerate the set

$$\{X = (X_1, X_2, \ldots, X_s) | \overline{f_{\mathscr{M}}}(X) = 1\}.$$

Formula (3) in Section then enables to compute the expected total number of possible byte modifications.

To summarize:

- as far as the AND detection function is concerned, when considering a detection pattern of size s, we have:

  $$\Delta = 256^s - 1.$$

  Consequently, short detection patterns are far better than longer ones. The latter yields more possible byte modifications, that may result in a non-detection;

- as far as other non-detection functions are concerned, evaluating Δ has a worst-case complexity in $\mathscr{O}(2^s)$. In this context, the copycat's effort increases with s. Long detection pattern are far better than short ones.

  The in-depth analysis of the antivirus that we have tested shows that detection pattern size are rather weak (about 15 bytes in average). Copycats' effort has consequently not to be very intense in practice, despite the overall complexity of Algorithm E-2. The parameter s should be larger ($s > 45$ at least).

Lastly, it is worth noticing that the weight of the non-detection function has a relatively high impact on the scheme resistance and on the parameter Δ. Since $\mathrm{wt}(\overline{f_{\mathscr{M}}}) + \mathrm{wt}(f_{\mathscr{M}}) = 2^s$, detection functions that we should consider in order to thwart copycat's action, are those that present a large weight. The experimental results that we have obtained with respect to the tested antivirus prove the latter to be rather weak when considering the resistance property.

Malware pattern efficiency

The black-box analysis revealed that the tested antivirus mostly used rather short malware detection pattern. As far as detection scheme efficiency is concerned, let us summarize our most important results and observations.

- Antivirus software can be divided into two groups, according to the average size s of the detection patterns:

  - short-size patterns (from a few bytes to a few tens of bytes). This group contains most of the commercial products;
  - long-size patterns (a few thousands of bytes). Only a few products belong to this group.

  We observed a trade-off between efficiency and resistance. As the resistance of the pattern grows, the number of false positive increases with it. This can be explained by the fact that detection function are AND functions. Other non-trivial functions should greatly enable to suppress this trade-off.

- The structure of the detection patterns (bytes involved, detection functions) presents striking similarities from one antivirus publisher to another (see Appendix 0.0.1). This proves that malware analyses are obviously a joint work within antivirus publishers community. Moreover, contrary to what they generally claim, sharing information does not, as it is, contribute to make products any different – with the notable exception of *Avast* and *AVG*. The most relevant case is that of *F-Secure* and *Kaspersky* who exactly share the same detection patterns. The structural properties of a PE file cannot justify alone the very close similarities among these products. Other explanations are to considered.

- Whenever possible, heuristic engine have been tested separately. At least for the *W32/Bagle* family, heuristic detection does generally not provide any significantly better detection. The difference with other traditional engine proved to be rather disappointing.

Secure Malware Pattern Scanning Schemes

In this section, we present a new scanning scheme designed to detect malware that strongly limits copycats' analyses. By using suitable combinatorial structures to describe malware detection patterns as well as probabilistic approach to manage them, we can then thwart pattern black-box analysis and extraction under reasonable security assumptions. Unless a relatively high number of other analysts collude, it is impossible for any copycat to extract all the pattern parameters (pattern size, pattern bytes, non-detection function). Any of his potential attempts to produce malware variants from previous ones will be inevitably doomed to failure, thus contributing to strongly limit malware spreading.

Combinatorial and Probabilistic Constructions

General description

Our main purpose is to consider a general malware detection pattern of s bytes (generally scattered over the malware code). Whenever the scanning engine is involved in a detection process, only a sub-pattern of size k is used under the following constraints:

- the value k must be relatively small compared to s;

- any sub-pattern is chosen at random;

- any sub-pattern is a fixed value which depends on the user/computer identification data;
- the whole detection pattern cannot be reconstructed unless we know (extract) at least τ sub-patterns;
- the number π of sub-patterns must be large as well as the value τ.

These constraints have been chosen in such a way to maximize the copycat's uncertainty and effort when facing the malware detection pattern extraction problem. In other words, he will not able to reconstruct the whole pattern except under conditions that result very difficult to implement in practice. Lastly, if he succeeds in producing any variant which becomes undetected with respect to a given sub-pattern, the probability to remain detected for other sub-patterns (other users) must remain high. The proliferation of the variant will be, as a result, extremely limited.

The main issue was to find suitable combinatorial objects exhibiting the required properties. The best candidates are undoubtedly combinatorial designs (Beth & al, 1999; Colbourn & Dinitz, 1996). Indeed, these objects show all the more interesting properties as their implementation only requires a limited amount of space/memory. The fourth constraint strongly suggested to consider (π, τ) threshold schemes (Menezes et & al, 1997). In our context, the secret to share is the whole detection pattern and each sub-pattern would then describe a share of the secret. Unfortunately, known threshold scheme contructions have been proposed when shares are numbers. In our case, the shares are more complex objects (typically collection of numbers). Some constructions have been proposed to extend classical threshold schemes to more complex objects like graphs (Kulesza & Kotulski, 2003). However, up to now, these extensions require too much memory/space. Thus, they are not suitable for commercial antivirus products.

Let us notice that the third constraint aims at helping computer crime investigation in order to trace and identify copycats who would be responsible for the production and spreading of new variants.

Technical description of the scheme

In this respect, two aspects must be considered: the combinatorial object which describes the pattern $\mathscr{S}_{\mathscr{M}}$ itself and the detection function $f_{\mathscr{M}}$ (and consequently $\overline{f_{\mathscr{M}}}$).

The combinatorial object .- We have considered $2-(s,k,\lambda)$ designs (known as *Balanced Incomplete Block Designs* as well or BIBD for short). For the sake of clarity, we will recall the definition of these objects as well as their most interesting properties.

Definition 3 *A* Balanced Incomplete Block Design *(BIBD) is a pair $(\mathscr{V},\mathscr{B})$ where $\mathscr{V}$ is a v-set and $\mathscr{B}$ is a collection of b k-subsets of $\mathscr{V}$ such that each element of $\mathscr{V}$ is contained in exactly r blocks and any 2-subset of $\mathscr{V}$ is contained in exactly λ blocks. The numbers v,b,r,k,λ are parameters of the BIBD.*

The following properties hold for a BIBD:

- *A BIBD exists if and only if $vr = bk$ and if $r(k-1) = \lambda(v-1)$.*

- $r = \frac{\lambda(v-1)}{k-1}$.
- $b = \frac{vr}{k}$.

An exhaustive presentation and classification of BIBDs as well as other combinatorial designs are provided in (Beth & al, 1999; Colbourn & Dinitz, 1996).

In our context, we have $\mathscr{S}_{\mathscr{M}} = \mathscr{V}$, $s = v$, $\pi = b = \frac{\lambda v(v-1)}{k(k-1)}$ and $\mathscr{B}$ describes the collection of sub-patterns.

The detection function .- We choose a Boolean function $f : \mathbb{F}_2^s \to \mathbb{F}_2$ as detection function. The DNF formula of f has weight 2^{s-1}. This weight value represents the maximal effort a copycat will face to extract the non-detection function (when considering the LOGICALMINIMIZE step) while maintaining a high level of pattern efficiency. According to the formalism developed in Section , the inputs of f are the k byte indices of the sub-pattern we consider whereas the other input variables are set to a constant value (projection). The resulting (projection) function with respect to the i-th sub-pattern is denoted f^i. One of the best choices, from an algebraic and combinatorial point of view, is to consider the following function:

$$f(X_1, X_2, \ldots, X_s) = X_1 \oplus X_2 \oplus \ldots X_s.$$

This function can be implemented in a very reduced way (see Section 0.0.1). Let us notice that from the copycat's point of view, the only way to retrieve this (simplified) algebraic normal form is from the DNF. This transformation has a general complexity of $\mathscr{O}(2^s)$. It goes without saying that not only many other detection functions could be chosen but also π different detection functions may be considered, one for each sub-pattern.

The scanning protocol .- During its installation, the antivirus sofware gathered some pieces of information on both the system and the user:

- the serial number of the processor (CPUID) and that of the hard disk (HDID);
- the MAC address (denoted `MACadr`);
- the user name `USRname` and his email address `@adr`;
- a secret value hidden in the antivirus software v.

Let us notice that other additional parameters can be considered.

Then, the sofware computes the sub-pattern index i as follows:

$$i = g(H(\text{CPUID} \oplus \text{HDID} \oplus \texttt{MACadr} \oplus \texttt{USRname} \oplus \texttt{@adr} \oplus v) \oplus \mathscr{N}).$$

The function H is a hash function whose input is the xor sum of the data encoded as integers while the function g outputs a random value which ranges from 1 to π, by means of a nonce $\mathscr{N}$. This value represents the indices of the detection sub-pattern that will be used for detection. The

function g is chosen in such a way that the value is fixed for a given user from one scanning to another. The purpose is two-fold. Firstly, it ensures that a user (or a copycat) will always use the same sub-pattern. Lastly, in case of computer crime investigation, the investigator will be able to prove whether a suspected copycat is involved or not in the building of a new variant.

During each detection process in which scanning is involved, the sofware uses only the i-th-sub-pattern along with the detection function f.

Mathematical Analysis

Let us now analyze our scanning scheme. In order to make things clearer, let us note $\mathscr{S}_{\mathscr{M}}$, the whole detection pattern (in other words the $2-(s,k,\lambda)$ design we consider) whereas $\mathscr{S}^i_{\mathscr{M}}$ will denote its i-th-sub-pattern and $f^i_{\mathscr{M}}$ the detection function taking the i-th-sub-pattern as input.

Scheme transinformation .- According to our notation and the definition of the scheme, we obviously have:

$$I(\mathscr{S}_{\mathscr{M},\mathscr{M}};\overline{f_{\mathscr{M}}},\mathscr{S}_{\mathscr{M}}) = H(\mathscr{S}^i{}_{\mathscr{M},\mathscr{M}}) << H(\mathscr{S}_{\mathscr{M},\mathscr{M}}) - H(\mathscr{S}_{\mathscr{M},\mathscr{M}}|f^i_{\mathscr{M}},\mathscr{S}^i_{\mathscr{M}}).$$

The black-box analyst manages to retrieve information on a very limited part of the detection pattern only.

Scheme resistance .- As previously stated, this property directly depends on both the weight of function wt($f^i_{\mathscr{M}}$) and k. For our scheme, we consequently have:

$$\mathscr{R}(\mathscr{S}^i_{\mathscr{M}},f^i_{\mathscr{M}}) = \frac{2^k-1}{k2^k} = \frac{1}{k} - \frac{1}{k2^k} = \mathscr{O}(\frac{1}{k}).$$

As a result, small values of k are better to get the best possible scheme resistance property (see Section).

Impact of copycat collusion .- We can imagine that a number of copycats try to collude in order to extract both the whole detection pattern $\mathscr{S}_{\mathscr{M}}$ and detection function $f_{\mathscr{M}}$. Let us determine how many copycats must collude for that purpose.

Proposition 1 *$\mathscr{S}_{\mathscr{M}}$ and $f_{\mathscr{M}}$ can be extracted with a collusion of at least $\tau = \lceil \frac{s}{k} \rceil$ analysts.*

Proof.
The proof is obvious since each block contains k points and since that $\tau \geq \lceil \frac{s}{k} \rceil$. This corresponds to the situation in which the blocks involved in the collusion have a null intersection. In this case, the design is said resolvable[6]. □

When τ copycats collude, they then have to face a detection pattern of size s. As far as the detection function DNF formula is concerned, we obtain the following result:

[6]In a *resolvable* BIBD, denoted RBIBD, the collection $\mathscr{B}$ can be partitioned into *parallel classes*. A *parallel class* is a set of blocks that partitions the point set $\mathscr{V}$. Two necessary conditions for BIBD to be resolvable are (1) $k|v$ and (2) $b \geq v+r-1$.

Proposition 2 *The detection function DNF formula extracted by a collusion of* $\tau = \lceil \frac{s}{k} \rceil$ *analysts contains at most* $\frac{s}{k}(2^{k-1})$ *terms.*

Proof.
The proof is obvious since the intersection of any collection of blocks does not contain any block of the design and since any two blocks may have a non-empty intersection. There is equality only when blocks involved in the collusion are paiwise disjoint (parallel class). □

This implies that even τ analysts would collude, the complexity of the LOGICALMINIMIZE step would be intractable and thus the non-detection function could be retrieved.

New variant probability of non-detection .- Let us now suppose that a copycat succeeds in extracting both $\mathscr{S}^i_{\mathscr{M}}$ and $f^i_{\mathscr{M}}$ from a detector that implements our scanning scheme. He can thus produce a new variant from a known, detected one. What is then the probability for this variant to remain undetected while it is "*in the wild*"?

Proposition 3 *The knowledge of both* $\mathscr{S}^i_{\mathscr{M}}$ *and* $f^i_{\mathscr{M}}$ *enables to produce a variant which is still detected with a probability* $P_{detection}$ *such that*

$$\frac{1}{2} \leq P_{detection} \leq 1.$$

Proof.
Assume that the copycat uses $\mathscr{S}^i_{\mathscr{M}}$ and $f^i_{\mathscr{M}}$ to produce his variants. Let us now suppose that this variant has spread throughout a computer whose detector selects the j-th sub-pattern. The modifications designed to produce the variant will affect the detection by means of the sub-pattern j with a probability equal to $\frac{1}{2}$ unless sub-patterns i and j have an empty intersection (detection probability is then equal to 1). Hence the result. □

This result shows that both the combinatorial object and the detection function must be carefully chosen. In particular, considering RBIBD is a more suitable choice than BIBD even if in this case, experiments proved that $P_{\text{detection}}$ is closer to 1 than $\frac{1}{2}$. Considering different detection functions, one for each sub-pattern is a better solution as well.

Implementation and Performance Results

The implementation of the proposed scheme is currently under investigation in our lab. We have considered different combinatorial objects. As far as security is concerned (especially the probability of non-detection for new variants), the best results have been obtained when considering parallel classes of RBIBD. Some interesting BIBD have also been satisfactorily used.

The upper bound of memory/space requirement is in $\mathscr{O}(\frac{s^2}{k})$ when considering a general $2-(s,k,\lambda)$ design (essentially due to the size of the incidence matrix describing the design). Consideration about phylogeny aspects (Goldberg et al., 1998; Karim Md et al., 2005) should help to

choose far better combinatorial designs which would enable several variants to be manage at once. As regards computing resources, we did not notice any significant increase in the scanning time.

Even though the implementation of our scheme need to be developed further – so that it may become as efficient current scanners – our results turn out to be very promising insofar as this scheme can be undoubtedly considered as practical solution.

Open Problems, Future Work and Conclusion

In the present paper, we have addressed both the problem about resistance against black-box analysis and classical detection scheme bypassing. We have first defined a new mathematical model for detection scheme that then enables us to define some properties that any reliable detection scheme should basically exhibit. Next, an in-depth analysis of a great deal of commonly used commercial antivirus products has been conducted with respect to this model and the relevant properties. The results of the experiments indicated that all the tested products turned out to be very weak. One of the obvious consequences of these investigations is that these existing weaknesses facilitate not only copycats'offences but also the spreading of variants derived from either original malware strains or previous variants.

We have presented a new scanning scheme that strongly prevents copycats'actions. Interestingly enough, not only it succeeds in highly limiting the spreadind of variants but also its implementation only requires few computer resources (memory and computing time).

Lastly, we have identified a number of open problems that should motivate further research:

- the classification and exploration of good detection functions $f_{\mathscr{M}}$;
- is it possible to find some structural properties of detection functions that would increase detection pattern efficiency while offering a high resistance level against black-box analysis. In particular, false positive probability could be significantly reduced thanks to detection functions which would be more appropriate than the AND function;
- find other – more suitable – combinatorial objects (Beth & al, 1999; Colbourn & Dinitz, 1996) offering more interesting properties than those we used in our scanning scheme (pairwise designs, parallel designs...).

We have just begun applying our model and approach to the behaviour monitoring detection techniques. In this new context, the concept of sequence-based detection has been replaced with that of the function-based detection and instead of considering detection patterns, we have, in the circumstances, considered malware behaviours. According to the very first investigations, there is every chance that our original model and approach may be fully transposable.

Acknowledgement

Many thanks to Boris Sharov from *DrWeb* and to the *G-Data* company who provide me with evaluation version of their antivirus.

Antivirus Software Analysis Results

We provide here the results of the black-box signature extraction performed on some *Bagle* variants and for most existing commercial antivirus software (Table 3). All the variants of the *W32/Bagle*

Products	Version	Viral Definition
Avast	4.6.691	0527-2
AVG	7.0.338	267.9.2/52
Bit Defender	7.2	09/16/05
DrWeb	4.33.2.12231	02/25/06 (104186)
eTrust	7.1.194	11.5.8400 (Vet)
eTrust	7.1.194	23.65.44 (InoculateIT)
F-Secure 2005	5.10-480	09/15/05
G-Data	AVK 16.0.3	KAV-6.818/BD-16.864
KAV Pro	5.0.383	09/19/05
McAfee 2006	-	DAT 4535
NOD 32	2.5	1.1189 (08/08/05)
Norton 2005	11.0.2.4	09/15/05
Panda Titanium 2006	5.01.02	01/17/06
Sophos	5.0.5 R2	3.98
Trend Office Scan	6.5 - 7.100	2.837.00

Table 3: Tested Antivirus Software (Versions & Viral Definitions)

family have been used as viral samples for the signature extraction algorithm E-1 which has been presented in Section . The naming convention for the viral variants is that used by VX Heavens (Vx Heavens), where we downloaded them in order to get a reference viral set.

Whenever several scanning engine were present, we have tested them separately when possible. The tables consider the best detection result.

Due to space limitations, the following tables give results for a few variants (exhaustive results are available upon request to the author). The table contains the indices of the bytes signature in the viral code but not the bytes themselves. As far as possible, the signature is given in full, if not, only a few beginning indices are provided in order to illustrate the “antivirus software phylogeny”.

Product Name	Signature Size (in bytes)	Signature (indices)
Avast	29	14,128 → 14,144, 14,146 → 14,157, 14,159
AVG	7,297	0 - 1 - 60 - 200 - 201 - 220 - 469...
Bit Defender	6	0 - 1 - 60 - 200 - 201 - 206
DrWeb	12	0 - 1 - 60 - 200 - 201 - 206 - 220... 222 - 461 - 465 - 469
eTrust/Vet	3,700	0 - 1 - 60 - 200 - 201 - 206 - ...
eTrust/InoculateIT	3,700	0 - 1 - 60 - 200 - 201 - 206 - ...
F-Secure 2005	11	0 - 1 - 60 - 200 - 201 - 206 220 - 240 - 241 - 461 - 469
G-Data	6	0 - 1 - 60 - 200 - 201 - 206
KAV Pro	11	The same as *F-Secure 2005*
McAfee 2006	7	0 - 1 - 60 - 200 - 201 - 206 - 220
NOD 32	7,449	0 - 1 - 60 - 200 - 201 - 204 - 205...
Norton 2005	0	(not an AND function)
Panda Tit. 2006	9	0 - 1 - 60 - 200 - 201 - 206 220 - 461 - 541
Sophos	28	0 - 1 - 60 - 200 - 201 - 206 - 220...
Trend Office Scan	7	0 - 1 - 60 - 200 - 201 - 206 - 220

Table 4: Viral Patterns for *W32/Bagle.A*

Product Name	Signature Size (in bytes)	Signature (indices)
Avast	9	8,162 - 8,166 - 8,170 - 8,173 - 8,175 8,180 - 8,181 - 8,187 - 8189
AVG	13,288	0 - 1 - 60 - 216 - 217 - 222 - 236...
Bit Defender	87	2,831 - 2,964 - ....
DrWeb	1,773	0 - 1 - 60 - 216 - 217 - 222 - 236...
eTrust/Vet	4,306	0 - 1 - 60 - 216 - 217 - 222 - ...
eTrust/InoculateIT	4,306	0 - 1 - 60 - 216 - 217 - 222 - ...
F-Secure 2005	105	0 - 1 - 60 - 216 - 217 - 222 - ...
G-Data	3	2831 - 2964 - 2965
KAV Pro	105	The same as *F-Secure 2005*
McAfee 2006	12,039	0 - 1 - 60 - 216 - 217 - 222 - ...
NOD 32	4,793	0 - 1 - 60 - 216 - 217 - 220 - 221...
Norton 2005	0	(not an AND function)
Panda Tit. 2006	37	0 - 1 - 59 - 60 - 216 - 217 - 222...
Sophos	15,028	0 - 1 - 2 - 4 - 8 - 12 - 13 - 16 - 24...
Trend Office Scan	146	0 - 1 - 60 - 216 - 217 - 222 - ...

Table 5: Viral Patterns for *Win32/Bagle.E*

Product Name	Signature Size (in bytes)	Signature (indices)
Avast	8	7,256 → 7,259 7,278 → 7,281
AVG	7,276	5739 - 5864 - 5866 - ...
Bit Defender	3,342	7,385 - 7,386 - ....
DrWeb	2,896	0 - 1 - 60 - 144 - 145 - 150 - 164...
eTrust/Vet	4,320	0 - 1 - 60 - 144 - 145 - 150 - 164...
eTrust/InoculateIT	1,311	0 - 1 - 60 - 144 - 145 - 150 - 164...
F-Secure 2005	3,128	0 - 1 - 60 - 144 - 145 - 150 - 164...
G-Data	2,954	0 - 1 - 60 - 144 - 145 - 150 - 445...
KAV Pro	3,128	The same as *F-Secure 2005*
McAfee 2006	8,084	0 - 1 - 60 - 144 - 145 - 150 - 164...
NOD 32	9,629	0 - 60 - 144 - 145 - 148 - 149 - ...
Norton 2005	8	0 - 1 - 60 - 144 - 145 150 - 164 - 445
Panda Tit. 2006	437	0 - 1 - 32 → 35 - 60 - 64...
Sophos	3,429	0 - 1 - 60 - 144 - 145 - 150 - 164...
Trend Office Scan	63	7,750 - ...

Table 6: Viral Patterns for *W32/Bagle.J*

Product Name	Signature Size (in bytes)	Signature (indices)
Avast	10	14,567 - 14,574 → 14,577 14,581 - 14,585 → 14,588
AVG	13,112	533 → 663 - 665 - ...
Bit Defender	6,093	0 - 1 - 60 - 128 - 129 - 134 - ...
DrWeb	6,707	0 - 1 - 60 - 128 - 129 - 132 - 133...
eTrust/Vet	4,343	0 - 1 - 60 - 128 - 129 - 134 - ...
eTrust/InoculateIT	1,276	0 - 1 - 60 - 128 - 129 - 134 - ...
F-Secure 2005	54	0 - 1 - 60 - 128 - 129 - 548 - ...
G-Data	53	0 - 1 - 60 - 128 - 129 - 548 - ...
KAV Pro	54	The same as *F-Secure 2005*
McAfee 2006	4,076	0 - 1 - 60 - 128 - 129 - 134 - ...
NOD 32	17,777	0 - 1 - 60 - 128 - 129 - 132 - 133...
Norton 2005	51	0 - 1 - 60 - 128 - 129 - 134 - 429 - ...
Panda Tit. 2006	1659	0 - 1 - 4 - 8 - 12 - 13 - 16 - 24...
Sophos	7,538	0 - 1 - 60 - 128 - 129 - 134 - 148...
Trend Office Scan	44	0 - 1 - 60 - 128 - 129 - ...

Table 7: Viral Patterns for *W32/Bagle.N*

Product Name	Signature Size (in bytes)	Signature (indices)
Avast	8	12,916 → 12,919 12,937 → 12,940
AVG	14,575	533 → 536 - 538 - ...
Bit Defender	8,330	0 - 1 - 60 - 128 - 129 - 134 - ...
DrWeb	6,169	0 - 1 - 60 - 128 - 129 - 134 - ...
eTrust/Vet	1,284	0 - 1 - 60 - 128 - 129 - 134 - ...
eTrust/InoculateIT	1,284	0 - 1 - 60 - 128 - 129 - 134 - ...
F-Secure 2005	59	0 - 1 - 60 - 128 - 129 - 546 - ...
G-Data	54	0 - 1 - 60 - 128 - 129 - 546 - ...
KAV Pro	59	The same as *F-Secure 2005*
McAfee 2006	12,1278	0 - 1 - 60 - 128 - 129 - 134 - ...
NOD 32	21,849	0 - 1 - 60 - 128 - 129 - 132 - 133 - ...
Norton 2005	6	0 - 1 - 60 - 128 - 129 - 134
Panda Tit. 2006	7,579	0 - 1 - 60 - 134 - 148 - 182 - 209...
Sophos	8,436	0 - 1 - 60 - 128 - 129 - 134 - 148...
Trend Office Scan	88	0 - 1 - 60 - 128 - 129 - ...

Table 8: Viral Patterns for *W32/Bagle.P*

Norton Bagle.E Non-Detection Function and Pattern

Whereas the extraction algorithm `E1` did not produce any AND detection function for Norton on some *W32/Bagle* variants, Algorithm E2 systematically extracted quite easily the relevant data (detection pattern $\mathscr{S}_{\mathscr{M},\mathscr{M}}$ and the non-detection function $\overline{f_{\mathscr{M}}}$. In order to prevent any misuse, the minimized form of $\overline{f_{\mathscr{M}}}$ will not be given (that is to say, we do not give the result after the LOGICALMINIMIZE procedure action). However, it is worth noticing that from a logic minimization complexity point of view, Norton viral non-detection functions are rather weak. They depend only on a limited number of variables (or equivalently bytes; around between 15 and 25 according to the variants) and terms. Moreover, the function DNFs are very sparse. This weakness enables the copycat to easily bypass Norton's scanner.

The viral pattern extraction algorithm `E2` of Section has produced the following non-detection function (before the logic minimization step) for *W32/Bagle.E* variant. The pattern depends on 15 bytes only, whose indices are

$$E = \{0, 1, 60, 61, 62, 63, 216, 217, 222, 223, 236, 237, 645, 646, 647, 648\}.$$

This result has been obtained once the COMBINATORIALMINIMIZE procedure action has occured. The notation x_i means that the byte of the pattern located at index i in the viral code, has not been modified whereas $\overline{x_i}$ means that it indeed has been.

Non-detection functions for other variants are available upon request to the author.

$$\overline{f_{\text{W32/Bagle.E}}} =$$

$$\overline{x_0}.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{646}.x_{646}.x_{647}.x_{648}$$

$$\vee\ \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ \overline{x_0}.\overline{x_1}.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}}.\overline{x_{648}}$$

$$\vee\ x_0.\overline{x_1}.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.\overline{x_{60}}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.\overline{x_{60}}.\overline{x_{61}}.\overline{x_{62}}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.\overline{x_{61}}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.\overline{x_{62}}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.\overline{x_{63}}.x_{216}.x_{217}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.\overline{x_{648}}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.\overline{x_{236}}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.\overline{x_{216}}.\overline{x_{217}}.\overline{x_{222}}.\overline{x_{223}}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.\overline{x_{217}}.x_{222}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.\overline{x_{222}}.x_{223}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.\overline{x_{223}}.x_{236}.x_{237}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.\overline{x_{236}}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.\overline{x_{237}}.x_{645}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.\overline{x_{646}}.\overline{x_{647}}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.\overline{x_{645}}.x_{646}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.x_{645}.\overline{x_{646}}.x_{647}.x_{648}$$

$$\vee\ x_0.x_1.x_{60}.x_{61}.x_{62}.x_{63}.x_{216}.x_{217}.x_{222}.x_{223}.x_{237}.x_{645}.\overline{x_{647}}.x_{648}$$

References

Angluin, D. (1988). Queries and Concept Learning. Machine Learning, Vol. 2-4, pp. 319–342.

Christodorescu, M., & Jha, S. (2004). Testing Malware Detectors. In *Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA'04)*.

Beth, T., Jungnickel, D., & Lenz, H. (1999). Design Theory. Cambridge University Press, ISBN 0-5214-4432-2 (Vol. 1) and ISBN 0-5217-7231-1 (Vol. 2).

Colbourn, C. J., & Dinitz, J. H. (1996). Handbook of Combinatorial Designs. CRC Press, ISBN 0-8493-8948-8.

Denis-Papin, M., Kaufmann, A. & Faure, R. (1963). Cours de calcul booléien appliqué. Coll. Bibliothèque de l'ingénieur électricien-mécanicien, Albin Michel éditeur.

Filiol, E. (2002). A New Statistical Testing for Symmetric Ciphers and Hash Functions. In: *Proceedings of the 4th International Conference on Information and Communication Security 2002*, Lecture Notes in Computer Science, Vol. 2513, pp. 342–353, Springer Verlag.

Filiol, E. (2005). Computer Viruses: from Theory to Applications. IRIS International Series, Springer Verlag, ISBN 2-287-23939-1.

Filiol, E. (2006). Advanced Computer Virology. IRIS International Series, Springer Verlag, to appear.

Fortinet, `http://www.fortinet.com/FortiGuardCenter/global_threat_stats.html`

Goldberg, L. A., Goldberg, P. W., Phillips, C. A. & Sorkin, G. B. (1998). Constructing computer virus phylogenies. Journal of Algorithms, vol. 26, pp. 188–208.

Kephart, J. O., & Arnold W. (1994). Automatic Extraction of computer Virus Signatures. In: *Proceedings of the 4th Virus Bulletin International Conference*, pp. 179–194, Virus Bulletin Ltd.

Karim, Md. E., Walenstein, A., & Lakhotia A. (2005). Malware Phylogeny Generation using Permutations of Code. Journal in Computer Virology, Vol. 1, 1-2, pp. 13–23.

Kearns, M., & Vazirani, U. (1994). An Introduction to Computational Learning Theory. MIT Press, Cambridge, MA, ISBN 0-262-11193-4.

Kulesza, K., & Kotulski, Z. (2003). On Secret Sharing for Graphs. `arxiv.org/cs.CR/0310052`.

McCluskey, E. J. (1956). Minimization of Boolean Functions. Bell System Tech. J., Vol. 35, Nr 5, pp. 1417–1444.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1997). Handbook of Applied Cryptography. CRC Press, ISBN 0-8493-8523-7.

Michaels, J. G. (2000). Algebraic Structures. In: *Handbook of Discrete and Combinatorial Mathematics*, Rosen K. H. (Ed.), pp. 344–354, CRC Press, ISBN 0-8493-0149-1.

Papadimitriou, C. H. (1995). Computational Complexity. Addison Wesley, ISBN 0-201-53082-1.

Quine, V. W. (1952). The Problem of Symplifying Truth Functions. The American Mathematical Monthly, Vol. 59, Nr. 8, pp. 521–531.

Quine, V. W. (1955). A Way to Simplify Truth Functions. The American Mathematical Monthly, Nov. 1955.

Shannon, C. E (1948). A Mathematical Theory of Communication. Bell Syst. Tech. J., Vol. 27, pp. 379–423 and 623–656.

Stimms, S., Potter, C., & Beard, A. (2004). 2004 Information Security Breaches Survey. UK Department of Trade and Industry, 2004. Available at `http://www.security-survey.gov.uk`. A video presenting the report to the press as well as a summary for decision-makers are also available on this website.

Szor, P. (2005). The Art of Computer Virus Research and Defense. Symantec Press and Addison Wesley, ISBN 9-780321-304544.

US Government Protection Profile (2005). Anti-virus Applications for Workstations in Basic Robustness Environments. Version 1.0, January 2005.
Available at `www.iatf.net/protection_profiles/index.cfm`

VX Heavens Database, `vx.netlux.org`.

Wegener, I. (1987). The complexity of Boolean functions. Wiley.

Modelling Information Assets for Security Risk Assessment in Industrial settings

Marcelo Masera & Igor Nai Fovino

Joint Research Centre

About Authors

Marcelo Masera is an Electronics & Electrical Engineer (1980), and an officer of the European Commission at the Joint Research Centre since November 2000.

He is in charge of the "Information Security of Critical Networked Infrastructures" area within the Institute for the Security and Protection of the Citizen. His interests are on the dependability and security of complex socio-technical systems, and specifically those related to critical infrastructures, large-scale systems-of-systems, information and communication technologies and the information society. He has published more than 60 papers in the fields of dependability, security and risk.

Contacts Details: Institute for the Protection and the Security of the Citizen, Joint Research Centre Via E. Fermi 1, I-21020 Ispra (VA), Italy. Phone: +39-0332-789238 email: marcelo.masera@jrc.it

Igor Nai Fovino received the M.S. degree in Computer Science with full marks in 2002 *and the Ph.D. with full marks in Computer Science in March 2006. He worked as at University of Milano in the field of privacy preserving data mining and computer security. In 2004 he was visiting researcher at CERIAS Research Centre (West-Lafayette, Indiana, USA) working on secure and survivable routing protocols. Currently he is researcher at the Joint Research Centre of the European Commission. He serves as reviewer for some international journals. His main research activities are related to the computer security and, more specifically, three are the main interests: System Survivability, Secure Protocols and Privacy Preserving Data Mining. In these contexts, in the last two years he published on international journals and conference proceedings more than 12 scientific papers.*

Contacts Details: Institute for the Protection and the Security of the Citizen, Joint Research Centre Via E. Fermi 1, I-21020 Ispra (VA), Italy. Phone: +39-0332-786541 email: igor.nai@jrc.it

Keywords

Information Assets, System modelling, Risk Assessment, Security Assessment

Modelling Information Assets for Security Risk Assessment in Industrial settings

Abstract

Industry has begun in the last years to take into consideration the use of Public Information Infrastructures (including the Internet) for remotely monitoring, managing and maintaining their technical systems. Concurrently, technical and business information systems are getting interconnected both through private and public networks. As a result, industry is exposed to internal and external cyber-threats, and the security assessment of the ICT infrastructures assumes a predominant relevance. However, underlying every useful security methodology there is a system description which decomposes the system in term of services, component, relationships and assets. In this paper, we focus our attention on a particular type of system asset to which, to our knowledge, the usual security assessment methodologies do not pay sufficient attention, the information asset. Such an asset, in fact, represents the core of every ICT infrastructure (commands sent to components are information assets, data stored into databases are information assets, data flowing through the network are information assets); therefore we believe that its proper description and analysis is key for assuring reliable results for security assessments. Starting from some classical definitions of information and knowledge, we examine this type of asset aiming at identifying the more suitable representation with respect to its security attributes. In more detail, we identify as interesting properties the interdependence between information assets, their life cycles, their dynamics (i.e. the flows of the information assets within the system), their topological location (in term of subsystems that hosts the information assets) and the correlation between the information assets and the vulnerabilities affecting the components of the system. We provide then a formal modelling framework for describing the characteristics of the information assets under a security assessment perspective.

Introduction

Security threats are one of the main problems of this computer-based era. A recent trend (Paukatong, 2005; Bowen, 2005; Fulton, 2005) shows that also in the industrial context, the use of ICT infrastructures has reached a level that impose taking in consideration cyber failures and attacks as one of the main threats to the security of such systems. More in details, industrial systems combine typical ICT components (e.g. data bases, communication protocols and networks, protocols, operating systems), with real-time elements (known as SCADA: supervisory control and data acquisition systems) where the control functions are implemented.

In such a scenario, it has become imperative for industrial actors to perform proper risk/vulnerability assessments, putting in evidence the main threats their critical systems are exposed to and eventually the effectiveness of the possible countermeasures. There exist in the scientific literature some interesting approaches to the risk assessment of ICT infrastructures [Alberts & Dorofee, 2002; Masera, Nai & Sgnaolin, 2005]. These methodologies have as core target the analysis of the system components, the interconnection between components and the set of correlated vulnerabilities, threats and attack mechanisms. However, they pay little attention to an important asset family of a system: the information assets. Although these methodologies have proven useful for zeroing in the security lacks of the analyzed systems, we believe that it is possible to improve the results of risk assessments by a more attentive and precise description of information assets.

Information permeates all aspects of industrial systems: based on the information stored and handled in a system, decisions are taken about its management and operation; information flows

within the system and through the connections with remote systems, link the different components and enable the command and observation of the system states; and both the discovery of vulnerabilities and the protection against threats heavily depend on the previous points.

It is therefore evident the necessity to consider the relevance of information assets to the failure of the system services, by means of their clear identification and characterisation, including the relevance of their security attributes, and a proper assessment of the related vulnerabilities and of the attack mechanisms that might assail them.

In this paper, we give a possible solution for this problem, presenting a framework for the description of the information assets of a target system, stating their relationships with the components of the system (at different level of details) and the vulnerabilities that can affect them. In order to do this we adopt as basis the risk assessment methodology presented in [Masera et al., 2005].

The paper is organized as follows: section 2 presents an overview of the risk assessment methodology. Section 3 presents a discussion of the characteristics of information assets under the risk assessment perspective. In section 4, our approach is presented and finally section 5 stresses the main conclusions.

The Security Risk Assessment Methodology: An Overview

The characterization of Information Assets (IA) strongly depends on the objectives the assessment process. In other words, an IA has several different interesting characteristics that can be modelled. However, only few of these characteristics are of interest for a particular scope. In our case, we are interested in the representation of the characteristics of an IA relevant to identify and evaluate the security of the system under analysis. It is, however, not sufficient to identify the characteristics to be modelled: in fact, different assessment methodologies put under analysis different aspects of a system, and thus require different types of models. For this reason, before describing the information assets, it is necessary to define the type of security risk assessment methodology for which the IA framework will be used.

As we have stated before, there exist in literature several types of risk assessment methodologies. We have chosen to adopt as reference the work of Masera et al. [Masera et al., 2005]. In such a work, the authors present a risk assessment methodology tailored to the analysis of the ICT infrastructure of complex industrial systems. In the remainder of this section, we give a brief overview of this methodology. More in details, this methodology foresees that in order to assess the security of a system, it is necessary firstly to provide a description of the system itself, of its components, of its assets, of the interaction and the relationships among the components, the assets and the external world.

Such a description (expressed analytically by tables) could be used to identify in a systematic way the vulnerabilities affecting the whole system. These vulnerabilities are then described by some significant parameters and used to identify the threat that can be associated to the components and to the whole system. From the analysis of this information, one can derive the evaluation of the possible damages to the components, their propagation to the system and the consequent attack pattern. All these operations are quantified in some risk related indexes that are then employed to perform the evaluation of the security failure risk and the countermeasures.

The approach adopted is based on five main steps. With regard to the topic of this paper, the information asset, are, in this methodology equated to any other type of asset relevant for the security of the system analyzed. Even if this can be a good choice in order to make easier the

analysis (i.e. it is not necessary to know a lot about the information asset, its flow etc.), we believe that a better and more detailed description of this particular asset, could give a great improvement in the analysis of a system.

Information Asset Characterization

As already claimed in the introduction, Information Assets have become extremely important elements of every technological system. Their relevance, however, is not due, as for other types of assets, only to their physical integrity or availability, but to a more subtle concept. In fact, if we imagine the IA as a black box in our system, the value, i.e. the relevance, of this black box is not associated to its physical attributes, but to the significance of the specific IA with reference to the particular state of the system at issue.

The significance of the data has therefore two facets: the value of the data per se, and the value of the data in light of possible security failures of the containing system (e.g. a system failure due to the lack of integrity or availability of the data).

In the first, the information asset connotes a type of knowledge whose contents and values are rather stable; in the second case, changes are relatively rapid. The approach to modelling has to fit both cases.

For this reason, before the characterization of Information Assets, we need to clarify the concept of information and the concept of knowledge contained by the information. Several definitions of Knowledge have been proposed in the scientific literature. As a first step, it is necessary to introduce the bricks by which Knowledge is built, data and information.

A data is a set of discrete, objective facts about events. In an organizational context, data is most usefully described as structured records of transactions. Drucker [Drucker 1999] said that information is "data endowed with relevance and purpose," which of course suggests that isolated data by themselves have no relevance or purpose.

With regard to the concept of information, in the scientific literature, some people equate it with meaning [Miller 1987]. Hearing a statement is not enough to make an event an informative act; its meaning must be perceived to make the statement informative. Arguing against this approach, Bar Hillel points out that "it is psychologically almost impossible not to make the shift from the one sense of information… i.e. information = signal sequence, to the other sense, information = what is expressed by the signal sequences" [Bar-Hillel 1955]. In another approach, information may be understood as "that which occurs within the mind upon the absorption of a message" [Pratt 1982]. A useful analogy is with the structure of systems that is viewed by some as being equivalent to information. Thus, "information is what remains after one abstracts from the material aspects of physical reality" [Resnikoff 1989]. However, a good general definition of information is given by Losee in [Losee 1997]: information is produced by all processes (in every context) and the values associated to the processes output is the information.

The difference between knowledge and information may be extremely thin. Epistemologists have strived to understand what it means to know something; in any case, we can think of knowledge as a set of information explicitly and implicitly correlated. From a formal point of view, Frawley, Shapiro and Matheus in [Frawley Shapiro & Matheus, 1992] give an interesting definition of Knowledge:

Definition 1: *Let F a set of facts, L a language, and C a measure of certainty, a pattern is a statement* $S \in L$ *that describes relationships among a subset FS of F with a certainty C, such that S*

is simpler (in some sense) than the enumeration of all facts in FS. A pattern that is interesting and certain enough is called knowledge.

We can deduce that the value of an Information Asset is represented by the value of the knowledge it contains. However, Frawley's definition was developed in the context of datamining processes and not in the context of system descriptions; the IAs have some characteristics that make the mapping operation between the two worlds not immediate. In order to release an instantiation of definition 1 into the system description world, there is the needs to clarify these characteristics.

First, from the point of view of their nature, IA appears under the form of measurements, commands, alarms, actions, policies, etc. The set of facts that are relevant for the significance of an IA varies for the different types of IA.

Secondly, from their function, information assets can be generated by human operators, by sensors or by the same industrial computing systems. In the analysis of an IA, from a security perspective, we need to take into account of this aspect.

Third, information assets exist within the protected zones of an industrial system, or are communicated through networks, either private or public.

Starting from what we presented previously, we are then now able to identify some relevant characteristics of an information asset:

(1) As of their nature, they are abstract. IAs are logical elements, implemented by means of software and hardware components, which encode and materialise them.

(2) Their value can be the results of some computation based on the value of other information assets.

(3) As objects, they are passive. They do not perform any action and they do not directly interact (we intend as interaction any type of operation in which an entity reacts to external stimulations or stimulates other entities) with any other entity. They can be assumed as simply token of information to be interpreted by components (e.g. a software) or human operator (they represent knowledge, and the other components interpret this knowledge giving it a meaning)

(4) They have a life–cycle: they are generated, undergo several handlings, are temporally stored, and finally are deleted. Even if a single asset represents always the same information, they can change form of representation during their life–time.

(5) They can flow through the system, and different instances of the same asset can be concurrently localised in more than one point of the system.

(6) In the case of aggregate information, they can be distributed in a sparse manner over the whole system.

From the security viewpoint, one has to consider that information assets are entities containing data, encoded in a representation that only acquires meaning in a given context. For this reason, it is not possible to directly associate attacks, vulnerabilities and exploits to an Information Asset per se, without consideration of the context of reference: associated components, time, state of the system, etc. In fact, roughly speaking, a vulnerability, by definition [Alhazmi , Malaiya and Ray (2005), Bishop and Bailey (1996)], is a weakness in the architecture design/implementation of an application or a service, and an information is neither an application (e.g. hardware, software) nor a service.

Although it is true that an information asset can be the target of a threat, the weaknesses used by attackers do not appertain to the information asset itself, but to the services and components implement, interact with and manage the information asset.

Most information assets are mobile objects; they flow from one component to another within and outside the system. The security evaluation has to take into account all the vulnerabilities present in all the possible logical paths the information asset can follow.

The "mobility" feature introduced before, has another important effect. In fact, as the information can flow through the system, it can be kept and maintained at the same time in different point of the same system. In the evaluation of undesirable effects of an attack/failure, it is then important to be able to model even the consistency of the IA, or, more specifically, the mechanisms by which the IA is maintained consistent over all its representations in the system.

Our thesis is that the characterisation of information assets for supporting security assessments entails four sets of knowledge:

- Patterns (following Frawley's meaning of the term) about the structural relationships among the information assets active in a given system.
- Patterns about the structural relationships between the information assets and the components of that system and their vulnerabilities.
- Patterns about the flows of information.
- Patterns about the dynamics of each information asset's lifecycle.

We call this collection of knowledge connoted by an information asset its **purport**. For instance, in the case of a *command* information asset, its **purport** includes all the "interesting and certain enough" facts conveyed or implied by it: e.g. the source, the target, the numeric value of the command variable, the range of validity in terms of time and system states, the set and order of components that interact with it and the vulnerabilities that might affect it.

Information Asset Description

Due to the fact that generally the system under analysis is very complex and extended, it is completely unrealistic to try to describe every type of information generated in the system. On the other hand, the name itself of what we want to describe (Information **Asset**), implies that we want to describe only information items that, for the analyst or for who had commissioned the analysis, encompass some relevant value. This consideration directly implies that the analyst must know thoroughly enough the system.

We assume that the analyst has already performed a system description in term of subsystems, services and components (in the adopted methodology [Masera et al., 2005] this corresponds to phase 1 (pre-assessment)).

Moreover, as the scope of our description is to use the obtained knowledge in order to make a security assessment, it is relevant to introduce from the beginning of the IA description some classical security properties that will be used in the remaining of this paper:

- *Confidentiality*: it is the partial or total concealment of information or resource [Bishop 2004] with respect to the rights of the entity that accesses the resource. In our case, given an information asset $\boldsymbol{ia_i} \in \boldsymbol{IA}$ where *IA* is the set of all the information asset of the system and considering an universe $\boldsymbol{U=\{u_1,\ldots, u_n\}}$ of all the entities which can in some way interact with ia_i , we define a confidentiality level assigned to u_k (i.e. the amount of information

contained by ia_i accessible from u_k) as a function $\boldsymbol{f}(\boldsymbol{ia}_i, \boldsymbol{u}_k) : \boldsymbol{IA} \times \boldsymbol{U} \rightarrow \aleph$, which gives as result an integer *n*. As it is possible to see, conceptually, the approach is very similar to the approach presented in the Bell-LaPadula work [Bell and LaPadula 1997].

- *Integrity*: it is generally defined as the trustworthiness of data or resources [Bishop 2004] and it is related to the task of preventing their unauthorized modification. In our case, we say that the integrity property of an information asset ia_i is preserved when only authorized users can modify it, and it is possible to identify any unauthorized modification. This presupposes the existence of some security mechanism. Given an asset $\boldsymbol{ia}_i \in \boldsymbol{IA}$ where *IA* is the set of all the information assets of the system, a set $\boldsymbol{U}=\{ \boldsymbol{u}_1,\ldots, \boldsymbol{u}_n \}$ of the entities which are able in some way to interact with $\boldsymbol{ia}_i$ and a set $\boldsymbol{O}=\{\boldsymbol{o}_1,\ldots, \boldsymbol{o}_k\}$ of all the operations which can modify an information asset, we can represent the mechanisms which avoid the unauthorized modification of an information asset by a function $\boldsymbol{f}(\boldsymbol{ia}, \boldsymbol{u}, \boldsymbol{o}) : \boldsymbol{IA} \times \boldsymbol{U} \times \boldsymbol{O} \rightarrow < 0,1 >$ where 0 means "no right to apply the modification" and 1 means "right granted". Moreover, considering that there exists the risk that an entity $\mathbf{u_j}$ claims to be $\mathbf{u_k}$ in order to gain the right to modify an asset, we represent at high level the mechanism that guarantees the identity of any entity $\boldsymbol{u}$ by a function $\boldsymbol{g}(\boldsymbol{u}, \boldsymbol{p}) : \boldsymbol{U} \times \boldsymbol{Evidences} \rightarrow < 0,1 >$, where ***Evidences*** is a set containing some evidences that can be used in order to proof the identity of $\boldsymbol{u}$, and <0,1> mean *false* and *true*. In the scientific literature there exist a large number of different security mechanisms that can be used in order to guarantee the integrity property, with different assurance levels. Therefore, there is the need to specify for each information asset the guaranteed minimum level of integrity. As in the case of confidentiality, we define such a level by a function $\boldsymbol{i}(\boldsymbol{ia}) : \boldsymbol{IA} \rightarrow \aleph$ which returns an integer *n* representing such required integrity level.

- *Availability:* it is related to the ability to have access to an information or resource [Bishop 2004]. In the case of information assets, we say that the availability property is preserved when accidental or malicious causes cannot prevent authorized entities to access the information asset. Each information asset has its own availability requirement. For example, for the access to an information asset $\boldsymbol{ia}_i$ a certain entity $\boldsymbol{u}_j$ may judge unacceptable a delay of 1 ms, while for another information asset $\boldsymbol{ia}_k$ the same entity $\boldsymbol{u}_j$ consider acceptable a delay of 10 min. Even the worst case of unavailability, that is the disruption of the information asset, can be represented as the case of a potentially infinite delay. The maximum access delay to an information asset ***ia*** which an entity ***u*** can tolerate can be then adopted as a measure of the availability level required. We can model the availability level as a function $\boldsymbol{al}(\boldsymbol{ai}, \boldsymbol{u}, \boldsymbol{k}) : \boldsymbol{IA} \times \boldsymbol{U} \times \boldsymbol{K} \rightarrow \aleph$ where ***K*** is a generic set of some additional knowledge (provided by the analyst) about the requirements of the entities contained in ***U***. The function ***al*** gives as output a maximum delay threshold, which is the maximum acceptable value for a given entity with respect to a given information asset, beyond which the IA must be considered unavailable. The greater such threshold, the weaker are the availability requirements.

As it is simple to imagine, not all the IA in a system require the same level of integrity, confidentiality and availability. This means that information about these properties should be linked to every information asset we want to describe.

Definition 2: we define as Security Requirements a tuple **SR=<Conf, Int, Av>**, where:

- *Conf* represents the weight (or relevance) of the confidentiality property for an IA.

- *Int* represents the weight (or relevance) of the integrity property for an IA.
- *Av* represents the weight (or relevance) of the availability property for an IA.

Keeping in mind the previous definitions and assumptions, we propose now a formal definition of information assets. In a preliminary analysis, and coherently with what we claimed in the previous section, we note that there exist two levels of description for an IA: a basic description, which is related to few, very evident characteristics of an IA (value, name etc.), and a more wide description, which we will call *Extended Information Asset*. This is the formal expression of the purport (as defined in the previous section) of an information asset, and it contains the knowledge implied by the IA in the context of its evolution in the system at issue.

The basic IA can be defined as follows:

Definition 3: an information asset is defined as a tuple $\mathbf{IA_i}$ **= <id, name, dsc, val, sr>**, where

- *id*: is the univocal identifier of the asset.
- *name*: is the name of the asset.
- *dsc*: is a free text description.
- *val*: is the perceived value of the asset.
- *sr*: is a security requirements tuple.

Definition 3 gives a very simple description of an IA at system level. However, this is not sufficient. In fact, let consider this example: in a system *S*, $\boldsymbol{IA} = \{\boldsymbol{ia}_1, \boldsymbol{ia}_2, \boldsymbol{ia}_3\}$ is the set of the IA judged relevant by the analyst. We know that the value of ia_3 is assigned by a human operator in light of the values of ia_1 and ia_2. In other words, we can think $\boldsymbol{ia}_3 = \boldsymbol{f}(\boldsymbol{ia}_1, \boldsymbol{ia}_2)$ where $\boldsymbol{f} : \boldsymbol{IA} \times \boldsymbol{IA} \rightarrow \boldsymbol{IA}$ is some logical or operational function. In this case, if anything happens at the lower levels of the system having a negative impact onto ia_1 or $ia2$, the effects will be propagated to ia_3. Therefore, it is necessary to represent the eventual relationships existing among the different IAs.

Definition 4: Let $IA=\{ia_1 \ldots ia_n\}$ be the set of the information assets of a system S, let *L* be a language, a statement $\boldsymbol{S} \in \boldsymbol{L}$ is an Information Asset Dependence (IAD) knowledge if it represents, with a relevance $\boldsymbol{c} > \boldsymbol{K}$ (where *K* is a lower bound chosen by the analyst), a relationship between two IAs (ia_b,ia_d).

Definition 4 gives a formal definition of the **IAD** knowledge. However, a similar definition is not useful if not supported by representation schema. Therefore, we introduce here the concept of IAD schema.

Definition 5: an IAD schema is an oriented graph with the following features:

- A set of nodes IAN where each node is a *basic IA* (see definition 3)
- A set of directed and weighted edges $L_{ia_j,ia_k} : L_{ia_j,ia_k} \in (\boldsymbol{IAN} \times \boldsymbol{IAN}), \boldsymbol{ia}_j \neq \boldsymbol{ia}_k)$, where each weight represents the strength of the relationship between ia_j and ia_k.

The previous definitions allow us to represent a "structural knowledge" related to the interconnection between different IAs.

As explained in the previous section, an important characteristic of the IAs is their mobility. In fact, they are able to flow through and outside the target system. This feature, in a context in which we are interested to study the impact of threats and failure on the IAs is very interesting. However, an important question regarding the IA flows is to choose at which level of details they must be represented. Even if it is true that an IA can flow from a component to another (where for

component we identify, as in the adopted methodology, every hardware and software object in the system), in a complex system an exhaustive approach describing all these flows results unfeasible. We choose then to represent the IA flows at subsystem level. This approach allows maintaining the description simpler and at the same time extremely flexible (in fact, the analyst in this way is free to decide in each singular case the size of the different subsystems and therefore describe the flows with different level of details).

To represent the IAs flows, there is the need to describe the subsystems contained in the system S under analysis and then to associate to every information asset the subsystems which in some way interact with them.

Figure 1 gives a high level logical overview of the structures described in this section, highlighting the links between information assets, subsystems and components.

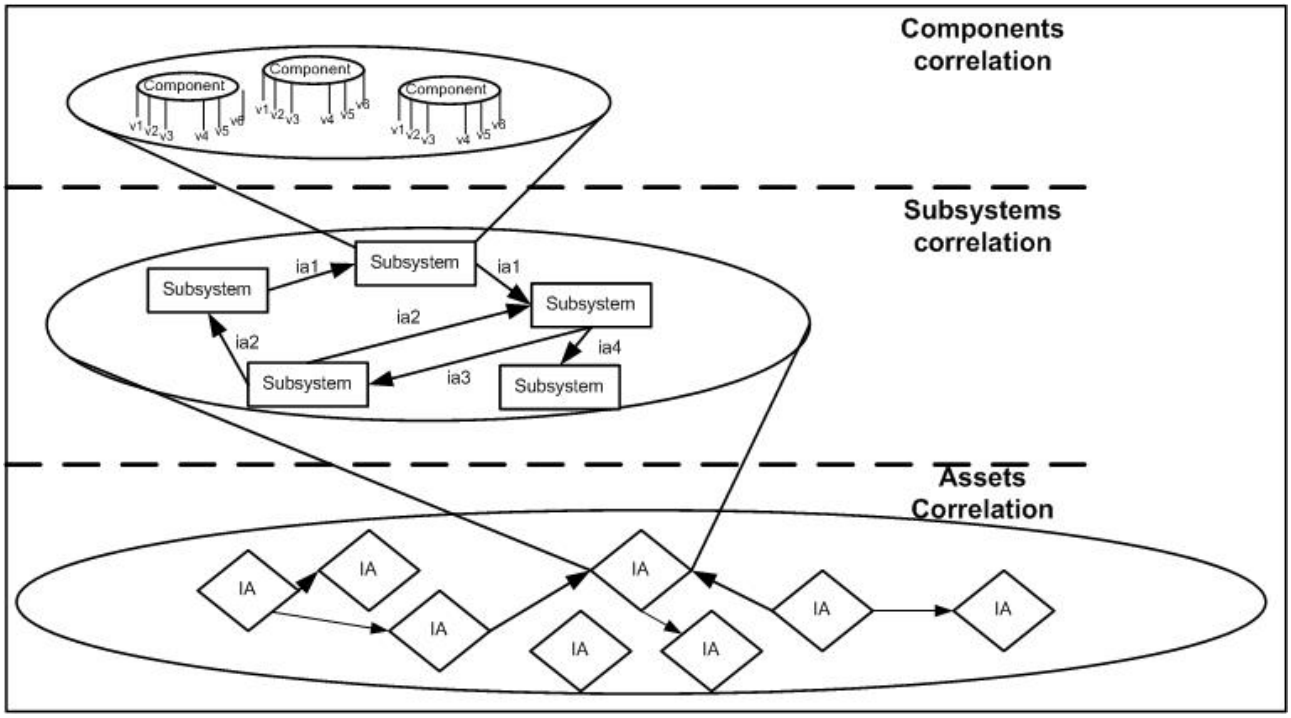


Figure 1: High level overview of the IA description

Definition 6: a subsystem s*b* is defined as a tuple $< \boldsymbol{id}, \boldsymbol{name}, \boldsymbol{description}, \boldsymbol{lia} >$ where:

- *Id* is the unique identifier for the subsystem
- *Name* is the name of the subsystem
- *Description* is a free text description of the subsystem
- *Lia* is a list of tuple $< \boldsymbol{ia}, \boldsymbol{sr} >$ where *ia* represents the id of an information asset which is managed or hosted by the subsystem *sb* and *sr* is a security requirement tuple magnifying the local security requirements for the information asset *ia*.

Definition 7: Let IA be the finite set of the information assets, we define the finite set of the subsystem through which an information asset ia_k flows, as a set $SB_k = \{sb \in SB \mid ia_k \in sb.lia, ia_k \in IA\}$.

It is interesting to note that the number of sets SB_k that can be described is equal to the size of the set IA of the information assets. We can then define $SB\_IA=<SB_1..SB_n>$ where $n=|IA|$ as the collection of these sets. Each element of *SB_IA* can be interpreted then as the description of the discrete relevant moments of an information asset life cycle. However, these sets do not capture any type of knowledge about the relationships of these moments. This type of knowledge can be described by the Information Asset Flow graph.

Definition 8: let $SB = \{sb_1 ... sb_n\}$ be the set of the subsystem of the system S, let $IA = \{ia_i ... ia_n\}$ be the set of all the information assets of the system, let $IA_i = \{ia \in IA \mid ia \in sb_i.lia, sb_i \in SB\}$ be the set of the information asset related in some way with the information asset sb_i and SB_IA be the collection of the sets SB_i as defined in definition 7, we define the Information Asset Flow (**IAF**) knowledge as an oriented, a-cyclic graph $IAF = (V, E)$ where:

- $V \subseteq SB$ is the set of nodes of *IAF* (it is the set of the subsystem having some relation with the IAs judged relevant in the previous phases of the description process)
- E is a set of oriented and labelled edges $L_{ia}(sb_k, sb_j) \in SB_{ia} \times SB_{ia}$, $SB_{ia} \in SB\_IA$, where the label *ia* means that the edge L is related to the flow of an asset $ia : ia \in IA_k, ia \in IA_j$.

In some cases, an information flow could be cyclic. This, however, introduces an additional layer of complexity to the IA description and analysis. How to solve this problem in an elegant way will be a scope of our future studies. However, as first solution we can transform the graph IAF into a new graph IAF` in which the edge between two nodes sb_i and sb_j which closes the cycle is removed and introducing a "virtual" information asset ia_v, which contains the original information asset, that flows from sb_i to sb_j.

The IAF graph represents the knowledge about the dynamics of the information assets of the system under analysis. Moreover, it can be used in order to represent the life-cycle of an information asset. In fact given the *IAF=(V,E)*, it is possible to identify the set of the information asset generators SBG, which is the set of the subsystem which generates the information assets and at the same way it is possible to identify the set SBS of the subsystem in which an information asset ends its life cycle. More formally, we define such sets as follows:

Definition 9: let $SB = \{sb_1 ... sb_n\}$ be the set of the subsystem of the system S $IA_i = \{ia \in IA \mid ia \in sb_i.lia, sb_i \in SB\}$ be the set of the information asset related in some way with the information asset sb_i **,** let IAF=(V,E) be the information asset flow graph of a target system S, we can define the set SBG of IA generator as $SBG = \{sb_i \mid sb_i \in SB, \exists L_{ia}(sb_i, sb_j) \in E : !\exists L_{ia}(sb_k, sb_i), ia \in IA_i\}$. At the same way, the set SBS the subsystem in which the IA end its life cycle can be defined as $SBS = \{sb_j \mid sb_j \in SB, \exists L_{ia}(sb_i, sb_j) \in E : !\exists L_{ia}(sb_i, sb_k), ia \in IA_i\}$

The structure **LC=<SB_IA, IAF, SBG,SBS>** represent the knowledge about the dynamics of the information assets and the knowledge about their life cycle.

Finally, in order to complete our vision of the information asset under the security perspective, we need to describe the exposition of IAs to failures and vulnerabilities. As we claimed in the previous

section, an information asset is a passive object; this implies that it cannot fail and it is not directly affected by vulnerabilities. However, the components of a subsystem can fail, and an attacker can exploit them taking advantage of their vulnerabilities. Then we guess that an information asset can be damaged by the failures and the vulnerabilities of the components of a subsystem that in some way can interact or manipulate them. In other words, an IA inherits the vulnerabilities and the failures of these components, that, in some way, compromise the security requirements associated with the IAs. The relation between the information assets, security requirements and vulnerabilities/failures is a very useful knowledge to be used in the risk assessment process. In order to represent this knowledge, we need at first instance to give a definition of component.

Definition 10: a component c_i is defined as a tuple $< \boldsymbol{name}, \boldsymbol{desc}, \boldsymbol{lov}, \boldsymbol{sb\_id} >$ where:

- *Name* is the name of a component
- *Desc* is a free text describing the component
- *Lov* is the list of the known vulnerabilities affecting the target component
- *Sb_id* is the id of the subsystem containing the component

It is important to underline that the component descriptions and the link between components and vulnerabilities is directly provided by the methodology presented in [Masera et al., 2005].

We need now to express the relation between components and information assets.

Definition 11: let $\boldsymbol{SB} = \{\boldsymbol{sb}_1 ... \boldsymbol{sb}_n\}$ be the set of the subsystem of the system S and *IA* be the set of all the information assets of the system, we define the set $\boldsymbol{IAC} = \{< \boldsymbol{ia}_i, \boldsymbol{c}_j, \boldsymbol{sr} > | \boldsymbol{ia}_i \in \boldsymbol{sb}_k.\boldsymbol{lia}, \boldsymbol{sb}_k.\boldsymbol{id} = \boldsymbol{c}_j.\boldsymbol{sb\_id}\}$ where *sr* is a local security requirement tuple, as the set representing the relationships between information assets and components and then indirectly, with the vulnerabilities affecting the components.

However, not all the vulnerabilities of a component *c* may be of interest for every related information asset. For this reason, the analyst needs to make a *pruning operation* on the space V of the vulnerabilities, removing from the list of vulnerabilities related to every component c, the vulnerability not interesting for a target information asset *ai*.

The result of this operation is a set representing the Information Asset Vulnerabilities knowledge (IAV)

Definition 12: let be IAC the information asset-components set, we define the Information Asset-Vulnerabilities set as:

$\boldsymbol{IAV} = \{<< \boldsymbol{ia}_i, \boldsymbol{c}_j, \boldsymbol{sr} >, \boldsymbol{list\_of\_vul} > | < \boldsymbol{ia}_i, \boldsymbol{c}_j, \boldsymbol{sr} > \in \boldsymbol{IAC}, \boldsymbol{list\_of\_vul} \subseteq \boldsymbol{c}_j.\boldsymbol{lov}\}$ where the subset *list_of_vul* contains only the vulnerabilities judged of interest by the analyst.

The set of the four knowledge patterns IAD, LC, IAC and IAV defined in the previous points, constitute the representation of the *Extended Information Asset (EIA)*, and of its purport forms the security viewpoint (bearing in mind that the concept of "purport" refers to the set of "interesting and certain enough" facts that correspond to the knowledge conveyed by the information asset).

As a conclusion, we state the purport of the *Extended Information Asset*, in brief *EIAP*, as the superset $\boldsymbol{EIAP} = \{\boldsymbol{IAD}, \boldsymbol{IAC}, \boldsymbol{LAV}, \boldsymbol{LC}\}$

Conclusion and Future Works

In this paper, we have proposed a characterization of the information assets, underlining the characteristics which are relevant in the context of the security analysis. Moreover, we have defined a formal description framework for information assets that can be synthesized in the extended information asset description that formalizes the concept of information asset purport, expressed as *EIAP = {IAD, IAC, IAV, LC}* where:

- IAD captures the concept of interdependence between information assets.
- IAC describes the set of security requirements of the information assets, with respect to the system components they interact with.
- IAV correlates the vulnerabilities of the system components with the information assets potentially damaged by their exploitations.
- LC represents the life-cycle of the information asset under consideration , and includes knowledge on the flows of the information assets (IAF).

This formal approach is being tried in our design framework (as represented in Figure 1) for supporting the analysis of security features. This approach allows the specification of the security requirements both at high level (by means of IAD) and at low level (by means of IAC), providing two security perspectives on the information asset: a global view and a local view.

As a part of future work, we plan to integrate the information asset description framework into the security assessment methodology we have developed [Masera et. al., 2005]. Moreover, in the context of such integration, we plan to elaborate synthetic indexes for measuring the assurance for the security attributes, to be computed with the elements of the model presented in the current paper. Finally, we plan to validate the integration of our security assessment framework and the information asset description presented here in some study cases.

References

Alberts, C., & Dorofee, A. (2002). Managing Information Security Risks: The OCTAVE (SM) Approach. July 2002, Addison Wesley Professional

Alhazmi, O., Malaiya, Y., & Ray, I. (2005). Security Vulnerabilities in Software Systems: A Quantitative Perspective. Lecture Notes in Computer Science, Volume 3654/2005. Publisher: Springer-Verlag GmbH.

Bar-Hillel, Y. (1955). An examination of information theory. Philosophy of Science, volume 22, pp.86-105

Bell, D. & LaPadula, D. (1997). The Bell-LaPadula Model. Journal of Computer Security, 5:303-339.

Bishop, M., & Bailey, D. (1996). A Critical Analysis of Vulnerability Taxonomies. CSE-96-11 September 1996

Bishop, M.(2004). Computer Security Art and Science, Addison Wesley, November 2004

Bowen, C.L., Buennemeyer, T.K., & Thomas, R.W.(2005). Next generation SCADA security: best practices and client puzzles. Systems, Man and Cybernetics (SMC) Information Assurance Workshop. Proceedings from the Sixth Annual IEEE 15-17 June 2005 Page(s):426 - 427

Drucker, P. (1999). Beyond the Information Revolution. The Atlantic Monthly.

Frawley, W., Piatetsky-Shapiro, G. & Matheus, C. (1992). Knowledge Discovery in Databases: An Overview. AI Magazine, pp. 213-228.

Fulton, B.(2005). Technical and administrative cyber security issues with implementation of a SCADA security upgrade. Security of Distributed Control Systems, 2005. The IEE Seminar on 2 Nov. 2005 Page(s):61 – 69

Losee, R. M. (1997). A Discipline Independent Definition of Information. Journal of the American Society for Information Science 48 (3), pp. 254-269.

Masera, M., Nai Fovino, I.,& Sgnaolin, R. (2005). A Framework for the Security Assessment of Remote Control Applications of Critical Infrastructure. ESReDA 29th Seminar.

Miller, G. L. (1987). Resonance, Information, and the Primacy of Process: Ancient Light on Modern Information and Communication Theory and Technology. PhD thesis, Library and Information Studies, Rutgers, New Brunswick, N.J., May 1987.

Paukatong, T.; SCADA Security: A New Concerning Issue of an In-house EGAT-SCADA. Transmission and Distribution Conference and Exhibition: Asia and Pacific, 2005 IEEE/PES 15-18 Aug. 2005 Page(s):1 - 5

Pratt, A. D. (1982). The Information of the Image. Ablex, Norwood, NJ, 1982.

Resnikoff, H. L. (1989). The Illusion of Reality. Springer-Verlag, New York.

Mystery Meat: Where does spam come from, and why does it matter?

Christopher Lueg, Jeff Huang & Michael B. Twidale

About Author(s)

Dr. sc. nat. Christopher Lueg is a Professor of Computer Science in the School of Computing at the University of Tasmania. Contact Details: c/o School of Computing, Private Bag 100, Hobart TAS 7001, Australia, e-mail christopher.lueg@utas.edu.au.

Jeff Huang is an M.S. Student in the Department of Computer Science at the University of Illinois Champaign, IL 61820 U.S.A., U.S.A. e-mail: huang6@uiuc.edu.

Michael B. Twidale, PhD is an Associate Professor in the Graduate School of Library and Information Science, University of Illinois, 501 E. Daniel Street, Champaign, IL 61820 U.S.A. e-mail: twidale@uiuc.edu

Keywords

email, spam, filtering, blocking,SMTP, digital divide, digital redlining,

Mystery Meat: Where does spam come from, and why does it matter?

Abstract

Unsolicited commercial email or spam is recognized as a problem disrupting email communication and costing the community dearly. In order to protect recipients from receiving spam, anti-spam measures building on technologies, such as filters and block lists, have been deployed widely. There is some evidence that certain anti-spam measures based on the purported origin of the spam cause unintended consequences which relate to issues of equity of access which we term digital redlining. Spammers have an interest in bypassing such measures by obscuring the real origin of their messages. Investigating these effects means we need to determine the true origin of spam, despite the efforts of spammers to confuse us and spam filters. The aim to find the true origin of spam is different from the objective of most anti-spam developers who are mainly interested in identifying spam when it knocks on their front door (mail server). In this paper we discuss why the difference between originator and delivery host matters when investigating digital redlining. We also highlight some of the difficulties we are facing when trying to determine the originating host as opposed to the delivering host.

Introduction

E-mail is widely regarded as one of the most important services provided by the Internet. In the U.S., for example, almost all Internet users use email which seems to be rapidly becoming more used than the telephone (Haythornthwaite and Wellman, 2002, p. 6). The term "spam" is used as a colloquial and convenient substitute for "unsolicited commercial email" (UCE). Spam has become a major problem causing significant costs to the community (e.g. McKusker, 2005).

In order to protect recipients from receiving spam, technical anti-spam measures have been deployed widely. Developing accurate anti-spam measures is technically challenging as there is no such thing as a precise, "technical" definition of spam; by definition, the nature of unsolicited/unwanted commercial email aka spam depends on the recipient's attitude towards receiving respective messages (Lueg, 2003; 2005). E.g., in a report on the spam problem and how it can be countered, the Australian National Office for the Information Economy defined spam as "unsolicited electronic messaging, regardless of its content" (NOIE, 2002 , p. 7). The report explicitly mentions that "arriving at an agreed definition of spam is a potentially contentious issue, as the direct marketing industry, ISPs, spammers, blacklisters and privacy and consumer groups have their own interests and views."

Spammers are effectively engaged in an arms race with spam recipients and providers of spam filtering technologies. Each measure by one side provokes a counter-measure by the other (e.g., viruslist.com; also see Hulten et al., 2004 for examples). However, innocent third parties can get caught in this crossfire, in particular perfectly legitimate email that gets mis-classified as spam and ignored or never even seen by its intended recipient. In this paper we focus on two related issues.

The first issue is that secondary effects of certain anti-spam mechanisms, in particular blacklisting of mail servers, may not only have the unfortunate effect of falsely declaring some email as spam, but in so doing disproportionately affect the emails of certain disempowered groups, an effect that we call digital redlining. Certain blocks of IP addresses may be redlined because mail servers operating in these net blocks have shown to be used for spamming and they are suspected to be used again in the near future. The most extreme case of digital redlining is blocking whole countries as proposed e.g., by SpamStopsHere. The interesting and ironic point is that there is evidence that

in certain cases, blacklists created in Western countries are redlining servers located in other –often economically disadvantaged– parts of the world due to spam that actually originated from those same Western countries. Rather than resolving the spam problem originating in their own countries, the Western blacklist filters discriminate against foreign servers that were (ab)used for sending spam around the world and back into the West.

The second issue we address is the problem of collecting supporting evidence for the above observations. The problem is that email headers can easily be manipulated which means it is difficult to extract reliable data about the originating host as opposed to the delivering host. This difficulty of determining the true origin and provenance of an email is of course both a challenge for research in this area, and may also contribute to designing more effective spam filters that can more equitably discriminate between legitimate and illegitimate email.

A related research challenge is that to our knowledge, there is no comprehensive conceptual framework available that could be used as a scaffold for the kinds of analyses we are interested in. Such a framework should support analyses of information equity issues in a systematic way. This lack of conceptual work had been noted by Lueg (2001) in the context of more general information distribution issues. In the related context of forensic computing, Slay, Turner & Hannan (2004) argue for establishing a framework supporting the use and also the positioning of different disciplinary approaches as this would help overcome the current focus of forensic computing on securing and analyzing the information "at hand", typically in a reactive way. In this sense, we see striking similarities between challenges in forensic computing and challenges in spam filtering research which appears to be focusing too much on immediate mail delivery issues and paying little attention to wider impacts of the technologies deployed.

Intended effects of anti-spam measures: protecting users and resources

The primary, intended effect of deploying anti-spam measures is a reduction of the amount of spam that end users have to deal with. Reducing the amount of net traffic devoted to passing around emails that recipients do not want to receive is also desirable, but end user attention is by far the most scarce resource and the one that should be optimized.

Most filtering targets the information contained in the body of a mail message (the part of a message users normally see) but can also consider information contained in the message header (mostly information used to transport the message but also From: and Subject: information). Typical examples of characteristics used to sort out spam messages are terms such as "free porn", "XXX", "Warez" or "Get rich quick". Many spam filters use a probabilistic approach known as Bayesian filtering, effectively awarding each email a number of ‘points’ for features that have been found to correlate with known spam, such as certain words in the content and headers, or a large proportion of HTML in the body. If the message’s points exceed a certain threshold, it is deemed to be spam. In this paper we focus on the header information and how it can be used in spam filtering, but of course in reality, this information is combined with the analysis of the body of the message to make an overall decision.

Spam filters may target the origin of a message. If a message is sent through a mail server that is known to be operated by a spam-friendly company, then this raises the probability that the message is itself spam, and for some filters this may be sufficient for the spam/notspam decision. Of course spammers are aware of the existence of spam filters and know much about their algorithms (certainly as much as is in the public domain, which is all the information that we have available for this analysis). Therefore spam messages may be designed to disguise their origin or provenance.

Related to origin-based filtering is the more radical approach of blocking. Blocking means a mail server simply refuses to accept any mail from certain servers, often according to blacklists shared on the Internet. Blocking approaches can use IP addresses, domain names and other information provided by the mail exchange.

Lists of alleged spam servers are shared on the Internet. Services such as the Spamhaus Block List allow mail servers to check in real time if a server trying to deliver email has earned a spammer reputation. The Spamhaus Project (2003), the organization hosting the above mentioned list, describes their block list as follows: "The Spamhaus Block List (SBL) is a real time database of IP addresses of static spam-sources, including known spammers, spam operations and spam support services."

Anti-spam products often implement combinations of different filtering, blocking and learning techniques (see Metz, 2003 for an overview). Kaspersky Anti-Spam ISP Edition, for example, uses a combination of linguistic analysis, formal analysis of message characteristics and blocking based on blacklists and whitelists.

Discussing the details of specific spam filtering technologies would exceed the scope of this paper. See e.g., the CEAS conference proceedings (URL http://www.ceas.cc) for further information.

It is also worth noting that in certain countries telecommunication legislation may prevent or restrict usage of specific spam filtering technologies. For example, preventing servers from receiving certain spam messages by blacklisting suspicious mailservers is basically illegal in EU countries because employees have the right to see these messages. See Gattiker (2005) for details.

Unintended consequences of anti-spam measures: digital redlining

The objective of using blocking techniques is to reduce a mail server's intake of messages likely to be classified as spam anyway. Messages may not be checked thoroughly since they may be assumed to be spam because they originate from alleged spam sources. This saves processing time, but of course means that such a coarse filter opens up the method to false positives.

Typically, blockings are based on information coming from blacklists shared on the Internet. There is anecdotal evidence that blocking not only reduces the spam intake but also may cause ‘collateral damage’ undermining reliability of email communication. The Spamhaus Project (2003) being a major host of such a black list notes the possibility of adverse effects as follows: *"Can the SBL block legitimate email? The SBL's primary objective is to avoid 'collateral damage' while blocking as much spam as possible. However, like any system used to filter email, the SBL has the potential to block items of legitimate email if they are sent from an IP under the control of a spammer or via IPs belonging to spam support services. The chances of legitimate email coming from such IPs are slim, but need to be acknowledged."*

From a technical point of view, blocking is an effective way to reduce a server's spam load. Anecdotal evidence suggests that blocking is also perceived as a way to suggest to mail server operators to protect their servers against misuse by spammers. A discussion in the newsgroup news.admin.net-abuse.blocklisting, comprising more than a hundred statements, indicates blocking is a double-edged sword though as there are quite different opinions regarding accountability and reliability of blocklists (see Blue, 2003 for details). There is an abundance of requests to be removed from block lists posted to the net-abuse related newsgroup news.admin.net-abuse.blocking (see, for example, Rodriguez, 2003). Often, these requests are coming from businesses renting IP blocks from major ISPs. Even though these businesses do not spam themselves, they are blocked

because their ISPs have a spam history or previous owners of their net blocks were spammers. Gaudin and Gaspar (2001) quote figures suggesting that using one particularly unreliable blacklisting service was found to cause 34% false positives, i.e., genuine email classified as spam. Cole (2003) provides a detailed overview as to why mail servers located in the net block (or IP range) of 'innocent' businesses may become 'collateral damage' and how these businesses may address the situation. Cohn & Newitz (2004) of the Electronic Frontier Foundation (EFF) discuss impacts of spam filters from the point of view of mailing list operators.

Varghese (2003a) reports the following incident: *"AOL says it is blocking email from Telstra's BigPond users because it has received complaints from its subscribers about spam being sent to them from BigPond addresses. Company spokesman Nicholas Graham said AOL had been [...] essentially compiling a whitelist of IPs from which mail would be allowed to reach AOL users."* BigPond is a major Australian ISP. This means AOL blocked a significant part of a whole continent's email users! Of course Telstra is a large company, and when it became a victim of this draconian anti-spam algorithm, it complained vociferously and was able to get the decision overturned within a week (Varghese, 2003b). Which leads to an interesting question of ethics and power in the digital world: are big, rich, powerful, western ISPs and servers better able to reverse presumably unfair blacklisting than their peers in poorer regions of the world? If so, the policy, which may appear solely to be about spam, and which all of the stakeholders may agree is a problem, may actually raise serious issues of equity across countries and levels economic and political power.

Hosted spam-filtering service SpamStopsHere mentions on their web site: "[i]t is well known that a huge amount of spam originates in China, Taiwan, Brazil and Argentina. It is clear to us that there are businesses in these countries that primarily just send spam. South Korea has a huge number of "open relays" which are computers that have been hacked and are controlled by spammers. Therefore, blocking email from countries notorious for sending spam is an effective filtering method. Which countries to block, if any, is a business decision. Click here for a discussion of why China, Taiwan and South Korea should be blocked." The idea that whole countries should be blocked because a lot of undesirable activity originates there (or appears to originate there – see later) raises many obvious questions about access and fairness. The similarities to the historic practices of certain US banks in discriminating in loan applications against areas of cities deemed poor economic risks (and not coincidentally frequently occupied by African Americans, so contributing to rising problems in those areas) inspired us to use the provocative term 'redlining'.

"Forensic" spam tracking challenges

Knowing where spam really comes from is useful in many kinds of approaches to prevent it, limit it or mitigate its consequences. As noted, the provenance of an email message can be a contributory piece of evidence in a judgment about its candidacy to be treated a spam by various filtering algorithms. Unfortunately spammers are aware of this and so take measures to try and conceal a message's true origins.

Although our interests as researchers in identifying spam are similar to those of anti-spam developers, our concerns are broader, extending to issues of the efficacy and equity of anti-spam measures undertaken. This requires gaining a richer picture of the true origins of spam. This is different from the objective of most anti-spam developers who are mainly interested in identifying spam at the time it knocks on their front door – their mail server. In terms of provenance, server's spam filters mostly focus on what is known about the last server to relay the message to them (see e.g., Goodman, 2004). Consequently, blacklists merely list prospective spam delivering hosts but do

not attempt to provide information about originating hosts.

Given the ease of fabricating the earlier history of the message's progress, this is an understandable simplification, but, we believe, an unfortunate one. In what follows we discuss why the difference between origin and delivery host matters when investigating digital redlining and highlight some of the difficulties we are facing when trying to determine the true originating spam host as opposed to the delivering host.

One of the challenges in the context of understanding digital redlining is identifying what header information is actually reliable. The way header information is forged is also of interest as it may tell who the spammer is trying to blame in the case of joe jobs. A 'joe job' means hiring a spammer to spam under the name of another person's domain, or web pages. The intended effect is that lots of people complain to the Internet service provider (ISP) hosting the domain or the web page advertised in the spam as they mistakenly assume they know the source of the spam. Further it is necessary to take into account information from other non-technical sources to understand how spamming is affecting the networked world. This means understanding spamming requires methods from a range of disciplines, from computer science and information science to social sciences and possibly political sciences. It also means it would be nice to have a conceptual framework supporting the use and the positioning of different disciplinary approaches as required. As mentioned earlier, a similar argument was put forward by Slay et al. (2004) in the context of forensic computing (McKemmish, 1999).

Identifying Fake Received Headers

The immediate technical challenge we are facing is that email is relayed from one server to another, and each server tacks on a Received: header indicating where it got the email from. These are simple unprotected text strings, listing the route the mail has taken, with the most recent server first. Thus it is possible for a spam-sending computer to falsely claim that it received the email from another computer, by inventing and adding prior Received: headers. These faked headers may be from real or nonexistent servers. Since there is no direct way of knowing whether or not a Received: header is legitimately added or not, it is generally accepted that there is no fool-proof method to find the origin of an email. What can be done instead is to identify the last external server that handles the email before passing it to an internal server, (Goodman, 2004). Assuming you can trust your internal servers, this IP cannot be falsified because it was reported by an internal server, which you have control over. Because of the lack of a fool-proof method to find the originating IP address of an email, the last external server is often used for tracking and reporting purposes instead. This may be good enough for spam filtering but not for analysis. So the origin of an email is generally not retrieved.

If a faked Received: header is found in the list, this indicates that one of the servers claiming to have subsequently received the message intentionally tried to foil attempts to identify the source of the email. Typically the guilty server is the actual originator the email.

For example imagine that we receive an email with header information including the following:

```
Received: from Echo <plus more info>
Received: from Delta <plus more info>
Received: from Charlie <plus more info>
Received: from Bravo <plus more info>
Received: from Alpha <plus more info>
```

The first line is the last stop on the way to us; Echo, our own internal server which we trust. If we can somehow determine that Bravo is a fake entry, and we trust Echo, then we must suspect that either Charlie or Delta are the true origin and that they faked Bravo (and also Alpha, which looks plausible to us).

Each received line usually includes ('from') the host name reported by the sending server through the HELO command, followed by a pair in parenthesis -- a host name gotten by the receiving server by performing a reverse lookup on the IP address, followed by the actual IP address of the sending server. Following this is the host name of the receiving server ('by'), and a ('with') token followed by some additional information about the SMTP server and an ID for tracking purposes, and finally, some time information (following a semi-colon).

```
Received: from relay7.cso.uiuc.edu (relay7.cso.uiuc.edu
[128.174.5.108]) by expms6.cites.uiuc.edu (MOS 3.4.8-GR) with
ESMTP id BDH13397; Sat, 21 Jan 2006 02:22:39 -0600 (CST)
```

In practice however, SMTP servers do not always use this format. It is possible that a forgery is done well enough that it is impossible to tell whether or not a header sequence is fake. Fortunately, many attempts to forge Received: headers are imperfect and can be detected. There are several ways a false Received header can be identified:

Illegal IP addresses

IP Addresses contain 4 bytes; each byte is a number between 0 and 255. Thus, any time a computer claims that it was passed email coming from an IP where a byte is greater than 255 is obviously false. Illegal IP Addresses are often generated by the programs that create the false headers, which simply string together 3 digits for each byte, so it is often possible to see an IP address such as 729.493.230.588.

IP addresses with leading zeros

A smarter program that generates random IP addresses will restrict the first digit to 0-2 and/or ensure that the second and third digits are 0-5 if the first digit is a 2. This will create valid IP addresses, but there can still be a clue that indicate the IP address was made by a program rather than one that was reported by an SMTP server. The giveaway is the leading zero in the IP address. For example, the IP address 4.235.08.96 would most likely have been generated by a program because an SMTP server would report it as 4.235.8.96.

Reserved IP addresses

There are several blocks of IP addresses which are not valid internet IP addresses. Most of these IPs are reserved for use by the IANA (Internet Assigned Numbers Authority), the organization that oversees allocation of IP address and documented in an RFC (RFC 3330). Blocking reserved IP addresses is a popular method of spam filtering, used by many spam filters including Spamassassin. These reserved IP addresses slowly change over time at IANA's discretion, usually because a certain block gets allocated to an entity or a regional internet registry, so to use this method to identify fake headers also requires information about when the email was sent as well. Fortunately, this time information is also in each Received lines.

Broken path

It is also possible to discover that a fake header has been inserted if there is a broken link in the chain of Received headers. For example,

```
Received: from exserver.chgh.org.tw ([203.69.196.131]) by
expansionpack.xtdnet.nl (8.11.6/8.9.3) with ESMTP id fA9LD9m04845
     for <foo@bar.com>; Fri, 9 Nov 2001 22:13:10 +0100

Received: from mcpeely.concentric.net (0-1pool5-38.nas2.los-
angeles1.ca.us.da.qwest.net[63.233.5.38])    by internation.co.uk
(8.11.0/8.9.3) with SMTP id gAJEYgl19984     for <foo@bar.com>;
Fri, 9 Nov 2001 20:29:03 GMT
```

Most likely, the second Received header is fake because there is no Received header to show how the email was moved from internation.co.uk to exserver.chgh.org.tw. Therefore, we suspect the second Received header to be fake and deem exserver.chgh.org.tw as the (spoofing) origin of the email. However, one must watch for legitimate mismatches caused by internal relaying (see below), but that is unlikely in this example.

Suspect Received Headers

Sometimes, even though it is impossible to tell definitively whether a Received header is fake, it might be possible to spot certain suspicious attributes that contribute to the probability that the header might have been tampered with. With an accumulation of such suspicious attributes, we may choose to make a determination that the header and so the whole message is fake, but we must be prepared to acknowledge that perfectly legitimate messages may also have these features.

HELO/EHLO mismatch

When a email client connects to an SMTP server, it is usually required to identify itself by its hostname using the HELO or EHLO (for extended SMTP) command. Originally, email clients were trusted to provide the correct hostname. However, now SMTP servers often also perform a reverse lookup on the IP address of the computer connecting to itself. Both the reported hostname name and then in parentheses the looked up name are reported in the Received header in the format below.

```
Received: from hostname.reported.by.helo.command.com
(actual.hostname.from.lookup.com [63.233.5.38]) ...
```

Theoretically, the hostname from the reverse lookup and the hostname given by the connecting client should be the same. However, since the hostname given by the HELO/EHLO command is up to the client, the hostname could be fictitious if the client wants to be evasive. Therefore, a mismatch between the hostnames could be seen as suspicious and headers beneath that header should be analyzed carefully. One caveat is that the hostnames could be different because of internal relaying and/or forwarding. For example, Microsoft hotmail might identify itself as mail.hotmail.com but the reverse lookup could report omc1-s7.bay6.hotmail.com.

SMTP path analysis, geographical path analysis

Using an IP to City database (e.g., geobytes, ip-2-location), it is possible to track down the geographical path of an email. Software that does this already exists (e.g., visualroute). It is possible to use this to analyze the path of the spam, and determine the rationality of the path of the email. If the Received headers of an email indicate that it has traveled from the United States to Belgium to

New Zealand, this is much more suspect than an email that has traveled only domestically, or only changed countries once. This will not determine the veracity of a Received header but merely indicates that a set of Received headers is suspect. It is possible for email messages to zig-zag their way across the world, exploiting variations in traffic density, but messages that oscillate between countries can also be a way of creating a spoof provenance.

Email that travels to multiple countries is very rare. The most complicated email path today occurs when an email is relayed from a set of sending servers to the destination set of servers. Thus, even if an email travels through different regions or cities, it is more suspect. It used to be fairly popular among US spammers though. For example:

```
Received: from ms1.hinet.net (root@168.95.4.10) by
amadeus.ifi.unizh.ch with SMTP; 7 Dec 2000 07:32:24 -0000
Received: from power.jyinfo.com.tw (root@[203.75.22.178]) by
ms1.hinet.net (8.8.8/8.8.8) with ESMTP id PAA26963; Thu, 7 Dec
2000 15:22:45 +0800 (CST)
From: ieowirut@centrum.cz
Received: from epost.de (1Cust41.tnt1.bloomington.il.da.uu.net
[63.27.139.41]) by power.jyinfo.com.tw (8.9.3/8.8.7) with SMTP id
PAA13339; Thu, 7 Dec 2000 15:18:53 +0800
```

Assuming the header lines are genuine, the email originates from a uu.net host in the US and was relayed to Switzerland (amadeus.ifi.unizh.ch) via Asian mail servers.

Dynamic hostnames in the path

Most SMTP servers have static IP addresses that do not change very often. On the other hand, a consumer's internet access, via say cable modem, often has a dynamically assigned IP address and respective hostname. Therefore, if a set of Received headers shows a dynamic hostname in the middle of the path, some of the headers may be fake.

Finding the "True" Originator

Algorithmically, the process of finding the originator of the email is fairly straightforward. Starting at the top of the set of Received headers, process down until either the end of the list, or a false or suspect header is found. However, as described earlier, determining what information is correct (or at least trustworthy) is a challenge and will remain a challenge even though authentication techniques are being introduced in a number of countries.

To gain an impression of trends in spamming we manually investigated a few hundred spam messages collected in December 2000, December 2001 and December 2005. The first two sets were collected in Zurich, Switzerland, the last set in Sydney, Australia.

December 2005		
straightforward	223	63%
dubious	46	13% (most relayed)

manipulated	80	23%
December 2001		
straightforward	48	42%
dubious	33	29% (most relayed)
manipulated	33	29% (most advertising.com spam)
December 2000		
straightforward	45	39% (incl. 8 surecom.com spams)
dubious	60	53% (most relayed)
manipulated	8	7%

Even though the data collection is not intended to be representative, the data does show a trend also reported in the literature: less third party relaying and more direct delivery. By direct delivery we mean that the delivering host is also the (alleged) originator:

```
Received: from haticeg (p169.net220148067.tnc.ne.jp
[220.148.67.169]) by filter.it.uts.edu.au (Postfix) with SMTP id
40214DF379; Thu, 1 Dec 2005 05:31:46 +1100 (EST)
```

The reason for less third party relaying is that most open relays were shut down (e.g., Hoffman, 2002) because of abuse by spammers. Direct delivery of spam points towards hosts that are infected by bots that are remotely controlled by a bot-master. As Kaspersky Lab state on their web site virus.com:

"Using open relay and open proxy servers is [...] time consuming and costly. First spammers need to write and maintain robots that search the Internet for vulnerable servers. Then the servers need to be penetrated. However, very often, after a few successful mailings, these servers will also be detected and blacklisted."

As a result, today most spammers prefer to create or purchase bot networks.

"In 2003 and 2004 spammers sent the majority of mailing from machines belonging to unsuspecting users. Spammers use malware to install Trojans on users' machines, leaving them open to remote use. [...] Anyone who has the client part of a program which controls the Trojan that has infected a victim machine controls the machine or network of victim machines. The resulting networks are called bot networks, and are sold and traded among spammers."

From the digital redlining point of view, the question will be where infected machines are located and whether there is a link between infection rate, blacklisting and economic strength of the respective geographic area. One might expect that computers in economically weaker regions are

easier to infect due to the cost of keeping software and hardware up-to-date. However, the most infected operating systems seem to be Windows 2000 and Windows XP systems i.e., relatively new and resource-intense operating systems more likely to be found in economically stronger regions than in economically weaker regions. Looking into blacklists we actually found a bias towards economically stronger regions and bot infections may explain the pattern.

It will also be interesting to see to what extent concepts, such as SPF or domain-based key authentication, will help ease the spam problem. Both are technologies that may help identify a forged sender. DomainKeys is an email authentication system that verifies the domain of an email sender and the message integrity. SPF is a system that allows domain name owners to use special DNS records to publicize which computers are authorized to send email from that domain. However, DomainKeys and SPF are not anti-spam technologies. They prevent spammers from spoofing addresses from domains that use the technology, but spammers can still spoof domains that don't publish SPF/DomainKeys or they can use their own domains. Also, even if there is a valid SPF or DomainKeys signature, it does not indicate that the message isn't spam. Also see URL http://security.weburb.dk/frame/show/news/3917 for a discussion of the economics of domain-based email authentication.

Summary and Future Research

In the long term, anti-spam measures deployed in Western countries to protect them against spam produced in these countries may contribute to excluding those located in economically less developed countries unable or unwilling to prevent spamming. As anti-spam legislation is getting tougher in Western countries, such as the U.S., E.U. and Australia, there is a good chance that spam will increasingly be sent from or relayed in developing countries. This paper provides three main contributions to the discussion of the impact of spam and anti-spam technologies.

First, we have chosen to focus on the provenance of spam; where it originates and the route it takes to get to the recipient. By contrast, the majority of analysis work by spam filter researchers has looked at the content of spam. Of course content is important, and perhaps is the most important part. But we believe that provenance can play at least a vital contributory role in understanding how spam and spammers operate and hence how best to develop measures to counteract it effectively and equitably.

Secondly we have noted that where provenance is used, due to the difficulty of obtaining accurate information, it is often used somewhat crudely in existing filters, focusing almost exclusively on the last hop of the message, and rendering that particular mail-bearer the subject of suspicion. This has clear consequences for efficiency and effectiveness of spam filters. However, far more importantly, we wish to raise the point that while providing considerable benefit, certain anti-spam measures, such as filtering and blocking, particularly when they use crude measures of provenance, can also create problems of access, specifically of equity of access where message from certain sources may be blocked excessively and unfairly more than from other, more privileged sources. We believe this risk of digital redlining needs more careful study.

Last but not least we see this work as a contribution to the discussion of the role of conceptual frameworks in information distribution settings and forensic computing. Frameworks should support the integration of different disciplinary methods ranging from computer science and information science to social sciences and possibly political sciences.

Acknowledgments

The authors are grateful to the anonymous reviewers for their insightful comments.

References

Blue, T.M. (2003). Blocklists Accountability, Standards, Who is Policing Blocklists. Posting to the Usenet newsgroup news.admin.net-abuse.blocklisting on 12 Aug 2003. Message-ID: <65ab995e.0308121542.4bccaede@ posting. google.com>

Cohn, C. & Newitz, A. (2004). Noncommercial Email Lists: Collateral Damage in the Fight Against Spam. Retrieved 30 January 2006 from URL http://www.eff.org/wp/?f=SpamCollateralDamage.html

Cole, W.K. (2003). Blacklists, Blocklists, DNSBL's, and survival: How to Survive as a Non-Combatant Emailer in the Spam Wars. A collection of frequently asked and too-often poorly answered questions. Retrieved 2 January 2004 from URL http://www.scconsult.com/bill/dnsblhelp.html.

Gattiker, U.E. (2005). Information Security This Week (EU-IST). Editor Urs E. Gattiker. Published 27 September 2005. ISSN1600-1869.

Gaudin, S. & Gaspar, S. (2001). The Spam Police. Tactics Used by Self-Appointed Spam Fighters Come Under Fire. Network World, 09/10/01 Retrieved 30 December 2003 from URL http://www.nwfusion.com/research/2001/0910feat.html.

Goodman, J. (2004). IP Addresses in Email Clients. Proceedings of the Conference on Email and Anti-Spam, July 2004.

Haythornthwaite, C. & Wellman, B. (2002). The Internet in Everyday Life. An Introduction. In: Wellman, B. and Haythornthwaite, C. (eds.) The Internet in Everyday Life. The Information Age Series. Blackwell Publishing, Malden, MA.

Hoffman, P. (2002). Allowing Relaying in SMTP: A Series of Surveys. Internet Mail Consortium Report: UBE-RELAY IMCR-016, http://www.imc.org/ube-relay.html

Hulten, G., Penta, A., Seshadrinathan, G. & Mishra, M. (2004). Trends in Spam Products and Methods. Clients. Proceedings of the Conference on Email and Anti-Spam 2004, July 2004.

Kaspersky (2003). Kaspersky Labs Now Battling Spam at the ISP Level. Product announcement released 12/30/2003.

Lueg, C. (2001). Towards a Framework for Analyzing Information-Level Online Activities. Proceedings of the 2nd Australian Information Warfare & Security Conference (IWAR 2001) Perth, WA, Australia, November 2001.

Lueg, C. (2003). On the Relevance of Spam and Anti-Spam Measures to Information Security Management. Proceedings of the 1st Australian Information Security Management Conference (InfoSECURITY 2003), Perth, WA, Australia, November 2003.

Lueg, C. (2005). From Spam Filtering to Information Retrieval and Back: Seeking Conceptual Foundations for Spam Filtering. 69th Annual Conference of the American Society for Information Science and Technology. Charlotte NC, USA, 28 October-2 November, 2005.

McKemmish, R. (1999). What is forensic computing? Aust. Institute of Criminology, Canberra.

McCusker, R. (2005). Spam: nuisance or menace, prevention or cure? Trends & issues in crime and criminal justice No. 294. March 2005. ISBN 0 642 53875 1; ISSN 0817-8542.

Metz, C. (2003). Corporate Antispam Tools. PC Magazine. Retrieved 16 February 2003 from URL http://www.pcmag.com/article2/ 0,4149,849390,00.asp.

NOIE (2002). Final Report of the Australian National Office for the Information Economy (NOIE) review of the spam problem and how it can be countered. Retrieved 3 May, 2003 from http://www.noie.gov.au/projects/confidence/Improving/Spam/ Interim_Report/contents.htm.

Rodriguez, G. (2003). SPEWS blocking a range with mine included , how to get out? Posting to the Usenet newsgroup news.admin.net-abuse.blocklisting on July 22 2003. Message-ID <b7efc7c3.0307220944.3e1a4d00@ posting.google.com>

RFC 3330. Special-Use IPv4 Addresses. IANA. September 2002. Retrieved 23 January, 2003 from http://www.ietf.org/rfc/rfc3330.txt.

Slay, J., Turner, P. & Hannan, M. (2004). Developing Forensic Computing Tools and Techniques within a holistic framework: an Australian Approach. 2004 IEEE Information Assurance Workshop, Westpoint, NY, June 10-12, 2004.

SpamStopsHere. http://www.spamstopshere.com/antispam_howitworks.aspx

The Spamhaus Project (2003). The Spamhaus Block List (SBL) Advisory FAQ. Retrieved 13 January, 2003 from URL http://www.spamhaus.org/sbl/sbl-faqs.lasso.

Varghese, S. (2003a). AOL blocking BigPond mail because of spam. The Sydney Morning Herald 29 April 2003.

Varghese, S. (2003b). Telstra problems with AOL resolved. The Sydney Morning Herald 30 April, 2003.

virus.com (n.d.) The Evolution of Spam. Retrieved 15 January 2006 from http://www.viruslist.com/en/spam/info?chapter=153350530

Spam Zombies from Outer Space

John Aycock & Nathan Friess
Department of Computer Science
University of Calgary

About the Authors

John Aycock is an assistant professor at the University of Calgary in the Department of Computer Science. He received a B.Sc. from the University of Calgary, and an M.Sc. and Ph.D. from the University of Victoria. He researches computer security and compilers, and conceived and taught the University's "Computer Viruses and Malware" and "Spam and Spyware" courses.

Nathan Friess is completing his B.Sc. in Computer Science at the University of Calgary. His research interests include computer security and wireless networking.

Mailing address: Department of Computer Science, University of Calgary, 2500 University Drive N.W., Calgary, Alberta, Canada T2N 1N4; Phone: +1 403 210 9409; E-mail: {aycock,friessn}@cpsc.ucalgary.ca.

Keywords

Zombies, botnets, spam, spyware, email worms

Spam Zombies from Outer Space

Abstract

How can better email worms be created? How can spyware claim more victims? How can better spam be sent? It turns out that these three questions are all related. We look at the future threat posed by email worms, spyware, and spammers that use zombie machines in a new way: sophisticated data mining of their victims' saved email. The end result skirts many defenses, and could trick even experienced users. We present new defenses that can be deployed now against this threat.

Introduction

In both advertising and social engineering, the common goal is to convince a targeted human to perform an action. The exact nature of the action varies with the domain: email worm authors may try to have the target run an executable attachment; spyware authors may want to direct the target to a specific web site for a drive-by download; spammers may want to have the target visit their web site or that of an affiliate. We will refer to email worm authors, spyware authors, and spammers using the generic term "adversary" since they share this common goal. In each case, the goal can be accomplished by sending out bulk email – spam – to potential targets.

Adversaries have one other goal, too. They want their spam to be as effective as possible. How can they do this? A flippant answer is for adversaries to use spelling checkers and both upper *and* lower case. A full answer is more complicated. The effectiveness of an adversary's spam can be characterized using the following formula:

$$effectiveness = \underbrace{sent \times (1 - P_{filtered})}_{\text{pre-delivery}} \times \underbrace{P_{read} \times P_{clickthrough}}_{\text{post-delivery}}$$

This formula measures effectiveness as a function of how many targeted people eventually perform the action that the adversary wants them to perform. The formula can be broken into two parts, one part describing what happens before the spam gets delivered to the target's mailbox, one part describing what happens post-delivery. The other terms are:

- *sent*, the amount of spam email messages sent by the adversary.
- $P_{filtered}$, the percentage of the spam that is intercepted *en route* by anti-spam defenses. The place where the filtering occurs is not relevant to effectiveness, and may be near the sending machine, the receiving machine, or done on a per-user basis.
- P_{read}, the percentage of the spam that is delivered to a potential target, and that is read by the target. By itself, this is a measure of the effectiveness of the From: and Subject: email headers used by the spam, as targeted users use the information in these fields (as displayed in their mail reader) to decide whether to read or delete the email.
- $P_{clickthrough}$, the percentage of targeted users that click on what the adversary supplies in their spam, such as a URL or an attachment. This describes how convincing the spam's body is.

The bulk of effort by adversaries, when trying to increase the effectiveness of their spam, has been directed at the pre-delivery part of the formula. Spammers increase the number of messages sent, to try and reach a wider audience of potential targets, or alter the messages in bizarre ways to try and evade filters. The latter tactic is particularly counterproductive, as it often renders the message unintelligible to the target.

Some efforts have been made at improving the post-delivery percentages, too. Prepending "Re:" onto a Subject: line to suggest that the spam is really a reply to some previous message was a clear attempt to improve P_{read}. As another example, the Mydoom.bb email worm tried to improve upon $P_{clickthrough}$ by using social engineering, and claiming to be from the IT staff (McAfee, 2005). These efforts are not very advanced yet, however. Ironically, if post-delivery attacks were more advanced, spam volume could actually *decrease* while yielding the same overall effectiveness.

In the remainder of this paper, we present a new method adversaries could use to improve spam effectiveness. The following sections explain the improved spam technique, the current defenses it avoids, our proof-of-concept implementation, and new defenses. We then look at further possibilities for this technique, along with related work.

Improving Spam

Improved spam will come from zombie machines. At first blush, this isn't terribly original, but the difference is in how the zombies are used. A zombie machine is not just a throwaway resource, a launching pad for DDoS attacks and spam; a zombie contains a wealth of data.

There are two key reasons why spam is suspicious to anti-spam filters and human targets alike. First, it often comes from an unrecognized source. Second, it doesn't *look* right. The evolution of spam zombies will change this. These new zombies will mine corpora of email they find on infected machines, using this data to automatically forge and send improved, convincing spam to others. In addition to the adversary, there are two other parties involved here: the *victim*, who owns a zombie machine, and whose saved email the adversary will be mining; the *target*, currently uninfected, that the adversary wants to click on something.

What features can be easily extracted from an email corpus? There are four categories:

1. Email addresses. The victim's email address and any other email aliases they have can be extracted, as can the email addresses of people with whom the victim corresponds. These addresses could be seen as fodder for harvesting by spammers, but we will show an alternate way to use these.
2. Configuration. This is information unrelated to individual messages, but related to the victim's email program and its configuration. For example, the User-Agent: email header added by the mail program to identify the sending program, the message encoding as text and/or HTML, and any automatically-appended signature file. The quoting style used by the victim's email program for replies and forwarded messages is a configuration issue too.
3. Vocabulary. The normal vocabulary used by the victim and the people with whom they correspond.
4. Email style. There are a number of distinctive stylistic traits, including:
    - Line length, as some people never break lines;[1]
    - Capitalization, or lack thereof;
    - Manually-added signatures, often the victim's name;
    - Abbreviations, e.g., "u" for "you";

[1] The email program and its configuration may also play a role here.

- Misspellings and typos;
- Inappropriate synonyms, e.g., “there” instead of “their”;
- Replying above or below quoted text in replies.

Email style should be correlated with the email recipient(s), because the victim may use both formal and informal email styles depending on whom they are sending to.

The adversary would begin by releasing malware that would infect machines and create an initial pool of zombie machines. The malware would mine any email corpora on the infected machines to determine as many of the victim-specific email features above as possible. Any of the features may change over time – a former co-worker the victim emailed may have left the company, or the victim may have switched email programs – so a prudent adversary would only use recent messages in a corpus.

Next, the adversary would supply a message to send out, with proper spelling, grammar, and capitalization. While the message could consist of multiple sentences, the adversary would supply it without line breaks. It is irrelevant whether the message was built in to the malware, or whether the adversary has a mechanism to communicate a fresh message to the zombie network. The malware would transform the message using simple filters, so that the message is rendered in the style the victim uses for a given target. For example, if Vic the victim normally uses an informal style to email Margaret the target, then the malware might determine the following parameters about Vic's prior email to Margaret:

Average line length	30 characters
Capitalization	none
Manually-added signature	“-vic”
Abbreviations	“u” for “you”, “r” for “are”

And the message supplied by the adversary:

```
Hey, have you seen this web site yet? It has some cool v
ideos: http://evil.example.com
```

Would be transformed into:

```
hey, have u seen this web site
yet? it has some cool videos:
http://evil.example.com

- vic
```

At this point, the malware could simply spam out the message to Margaret and other targets. The transformed message still looks “spammy” by itself, however, even though it will come in the correct style from a recognized source. We think that more effective spam will not be sent like this, but in one of the following forms:

- Replies to incoming email. Here, the malware monitors legitimate incoming email addressed to the victim. When a new message arrives, its sender is targeted; before the victim has a chance to reply to the email, the malware composes and sends a reply in the victim's style:

```
Sorry, I won't be able to get to your email for
a while.  However, this should keep you amused
in the meantime: http://evil.example.com
```

This type of reply is consistent with observed human behavior in email interactions (Tyler & Tang, 2003). The deception can be further extended if the malware doesn't generate a reply when the victim is actively using the infected computer, because the victim may actually be replying to the email.

There are other advantages for the adversary. The malware is able to quote the incoming message, lending more credence to the spam. Also, by replying, the malware has a valid Message-Id: to use in the In-Reply-To: and References: email headers – a target using an email program that shows threads will link the malware's reply to the original mail.

- Forwarded email. The forwarding of messages, attachments, and links is a common occurrence in email, and is arguably a form of gift-giving (Smith, Ubois, & Gross, 2005). However, forwarding by humans is done bearing in mind the recipient's interests (Smith et al., 2005). A target's interests are not only hard for malware to automatically determine, but they are highly unlikely to coincide with the spam the adversary has to send.

  Effective spam can shy away from the problem of targeting specific interests by using humor, a common denominator. An adversary can lure targets with the promise of funny pictures, sound clips, or animations; these can be easily copied from the Internet onto an adversary's web site (which also sports a drive-by download) or attached to spam. Having such a real lure acts as a Trojan horse, and can delay suspicion by the target.

  The other problem is making the forwarded email come from an authentic-looking source. Malware can mine information from the victim's email corpora about groups of people that the victim usually CCs or sends email to; this is more precise targeting than using a victim's address book, because it reflects the victim's actual, current email practice. This information can be used for several purposes:

  - Sending supposedly-forwarded spam from the victim to an authentic social group of targets;
  - Making reference to a common acquaintance in the spam;
  - Spamming fake “e-card” announcements from the victim to a plausible group of recipients.

  As a special case, some people with a close, informal relationship simply forward URLs in the Subject: line, with a blank message body. The malware can look for these in the victim's email corpus, and exploit the trusted relationship to send URLs to the target similarly.

None of these schemes will work if the malware sends multiple copies of the spam to a target, or if the target receives one of these spams for every message they send to the victim. The malware must keep a database of spam attempts, and only send one spam per target. Further suspicion can be avoided if spam is not sent to already-infected machines; a zombie can covertly hide an infection indicator in outbound email with some crude steganography (Wayner, 2002). For example, the last line of all email from a zombie machine (whether a spam message or not) may have thirteen spaces appended onto it.

Malware can also avoid sending email at unusual times. As with other features, a victim's email corpus can be mined to determine those times when they usually respond to email. This also contributes to making the spam look more normal.

The path traveled by more effective spam is worth noting. A topologically-aware worm (Staniford, Paxson, & Weaver, 2002) is a worm that scans using information about network topology, to avoid the wasted effort in random or quasi-random scanning. More effective spam travels along the topology of *social* networks instead of computer networks, like an IM worm that only tries to infect people on "buddy lists". The "small world" hypothesis holds that any two people are separated from each other by a short chain of acquaintances (Milgram, 1967; Travers & Milgram, 1969); this idea has recently been applied to searching in computer networks (e.g., Simsek & Jensen, 2005). One theory is that a social network works this way due to the presence of some highly-connected people (Gladwell, 2002) scattered throughout the network – intuitively, some people are more popular than others. This would mean that more effective spam only has to fool a handful of targets, enough to turn a highly-connected target into a victim, in order to be widely dispersed.

Defenses Avoided

The more effective spam described in the last section avoids many defensive measures that targets may have in place. Here, we look at anti-spam defenses responsible for $P_{filtered}$; detecting any malicious content run or downloaded as a result of the victim clicking what the adversary supplies is a separate issue, and is beyond the scope of this paper. For each defense, we give a brief description of it and an explanation of how more effective spam can evade it. Specific references are given where appropriate; Zdziarski (2005) is a good general introduction to the area.

- Whitelisting. Incoming email is allowed only from those senders who are on a special list, called a "whitelist". Effective spam is sent from the victim, *as* the victim, to the target; if the victim is not whitelisted already, then no legitimate email from the victim can get through to the target.
- Blacklisting. Mail is refused if the sender is on a special "blacklist". Again, if the victim is blacklisted, then legitimate mail from the victim could not be getting through to the target, and thus blacklisting is ineffective here.
- Greylisting. Mail from first-time senders is initially refused, on the premise that spammers won't retry transmission of email, but legitimate senders will. Senders that do retry are temporarily whitelisted to prevent future retransmission delays (Levine, 2005). More effective spam can piggyback on the victim's normal email system, which (presumably) handles retries properly.
- Rate limiting. Rate limiters, or throttles, assume that spam is being sent in large quantities, and slow the rate of outbound email to suppress spam (Williamson, 2003). This assumption is not valid for more effective spam. Reply-based spam will operate only as fast as legitimate incoming email to the victim; forward-based spam will not be convincing if sent in large quantities, so a smart adversary would limit the rate of forward-based spam themselves.
- Tarpits. A tarpit is a special mail transport program that replies… very… slowly… to incoming spam transmissions, to slow down overall spam transmission or at least annoy spammers. There would be no way to apply a tarpit against more effective spam without harming the victim's legitimate email, and again, this defense assumes large quantities of spam transmission.

- Spam traps. Spam traps are email addresses made public (e.g., posted to a web page) with the intention that they be harvested by spammers. A spam trap does not correspond to a real person, however, so any email sent to that address must be spam. More effective spam is sent to addresses from legitimate incoming email, or to legitimate addresses mined from a victim's saved email, thus avoiding spam traps.
- Challenge/response systems. A first-time sender is issued a challenge, which the sender must reply to prior to their email being delivered. Essentially, the goal is to try and prove that the sender is a human being. A successful response to a challenge results in the sender being whitelisted. Our reply-based spam would not be subject to a challenge; correctly-designed challenge/response systems would automatically whitelist the victim, because the target initiated the email exchange. Forward-based spam is sent to people named in the victim's saved email, for which the victim would have already replied to the challenge.
- Proof of work. Proof-of-work systems have the sender demonstrate that they have spent some nontrivial amount of computer time computing a value (Dwork & Naor, 1993). The idea is that this would impede spammers from sending mass email by imposing a tangible "cost" on them. Yet again, this defense assumes large volumes of spam, and is inapplicable to more effective spam.
- Authentication systems. Mail is verified to be sent from a legitimate mail sender for the domain stated in the email (e.g., Sender Policy Framework, Wong & Schlitt, 2005), *or* mail is verified to be sent by the stated domain and unaltered in transmission (e.g., DomainKeys, Delany, 2005). More effective spam is sent from the victim's machine in an authorized manner, and is unaltered *en route*, so sender authentication does not help as a defense.
- Checksum-based filtering. A collaborative scheme whereby a checksum, preferably one insensitive to minor text alterations, is computed for each incoming message. The checksum is sent to a central server that stores the checksums of known, reported spam; a hit from the server indicates that the email is spam. More effective reply-based spam would not be vulnerable to detection, because the target's quoted text in the reply would throw the checksum off. Forward-based spam could be caught, if the adversary's malware only introduces minor variations to the spam. However, the low-volume, deceptive nature of more effective spam may result in either the spam (and its checksum) not being reported, or it falling short of a "bulk email" threshold used by the central checksum server.
- Statistical filtering. Statistical filters, such as Bayesian filters (Graham, 2002), predict the likelihood of email being spam or ham based on how often the words[2] in the email appeared in corpora of previous spam and ham. Reply-based spam includes the target's quoted text to push the email in the "ham" direction. In a sense, this is a sophisticated form of "word salad", where spammers try to slip by statistical filters by including random words or phrases in their spam (Graham-Cumming, 2004). Forward-based spam is more vulnerable to detection, though, because its message is more exposed.
- Pattern-based and heuristic filtering. Pattern-based filters detect spam by looking for a match in a predetermined set of "this is spam" patterns. Heuristic filters combine a large number of heuristic tests (which may include all of the above types of defense) to make a spam/ham judgment about email. Both cases are similar to statistical filtering: reply-based

[2] "Word" is used in a very loose sense here.

spam is likely to avoid detection, forward-based spam may or may not be caught. Because more effective spam deceives by looking like legitimate email, the risk of trying to increase an anti-spam filter's detection rate is an increased number of false positives.

Of course, there are many variations and combinations of these defenses; this should be taken as an exhausting, but not exhaustive, list. The main point is that existing defenses are not enough to combat more effective spam.

Proof-of-Concept Implementation

We performed some experiments mining data out of email corpora, to determine how easy it would be to mount our proposed spam attack, and to see how convincing the result would be. Although they could be forged by the adversary's malware, a number of the features mentioned previously will be automatically incorporated into outgoing mail if the malware uses the victim's email program to send its mail, including From:, User-Agent:, preferred encoding (text and/or HTML), and any automatic signature file.

For our experiments, we focused on mining capitalization, commonly used abbreviations, common misspellings, manually entered signatures, and social groups from two corpora. The first corpus we used was a small dataset that we manually generated ourselves; the second was the public corpus of Enron email (CALO Project, 2004). In each of the cases, except for the social groups, we only examined the victim's outgoing email folder so that we are only examining text that the victim actually wrote. The implementation was comprised of several Perl scripts, each of which mined one of the listed features from the corpus. Each script did the following for each email message:

1. Strip off the automatically-appended signature, if any. This signature was found by searching the email program's settings.
2. Search for and remove any quoted text, in case this message was a reply or forward.
3. Perform the data mining (e.g., capitalization, manual signature).
4. Save the statistically significant results for later use in generating spam.

When mining the victim's capitalization habits, the script merely split the text into individual words, counting the number of words that were completely lower case, completely upper case, or mixed case. The decision as to which form of capitalization the user used for any given message was based on a percentage of which of the three forms were encountered. Although we simply estimated that 20% of words should be capitalized before we declare that the victim was using mixed case, a proper analysis of English text could be used to fine-tune this number. Once the script had processed all of the emails, our spam generator mimicked the capitalization most used by the victim.

Mining abbreviations was quite similar to capitalization, except that our script started with a list of commonly used abbreviations such as:

u	you
r	are
ur	your
cya	see you
ic	I see
afaik	as far as I know
thru	through

Each time one of the abbreviated forms of a word was encountered, the script incremented a counter for that type of abbreviation. In the end, if we found more than five occurrences of an abbreviation, our generator also used that abbreviation, although this threshold could be fine-tuned as well.

The script for mining common misspellings sent each word to a spell checker (Aspell), and noted which words were flagged as incorrect, as well as the suggested replacement. Again, if we found more than five occurrences of a spelling mistake, and the spell checker provided a suggested replacement word, then our generator performed the reverse replacement when creating the spam email.

When mining manually entered signatures, rather than looking at each word of the text, the script examined the last non-blank line of the text. The script kept track of each unique signature, and how many times it was encountered throughout the corpus. Similar to the other transformations, if a signature was found more than five times, it was added to the pool of signatures that were randomly added to generated spam.

The script that mined social groups was slightly different in that it examined all of the headers of an email, as well as the entire body, including quoted text. The script built a list of all of the email addresses found in the message for two purposes. The first reason was to count the number of times any given pair of addresses appeared in the same message. The second was to create a map between email addresses and real names of individuals. Both of these results were also used by our spam generator.

Finally, our spam generator used all of the above results to transform any given text into a spam email. The input for the generator was an email to reply to and some text to add to the reply, which would presumably contain a URL for our target to click on. The generator then output an email message that could be sent to the target.

The results of our experimentation were very positive (for the adversary). For the Enron corpus, our scripts were able to mine almost all of the mentioned transformations successfully. In one mailbox, we found over 670 occurrences of the abbreviation “r” and over 160 occurrences of “u”. Furthermore, even though the formatting of the emails was quite erratic, our scripts also successfully found the victim's manual signature “Vince”, with 1470 occurrences, far above the next best guess at 216 occurrences. Mining common spelling mistakes was less successful, as the spell checker had a tendency to correct things like people's names when it shouldn't have. This result is not entirely surprising. However, the spell checker did find minor nits such as “p.m.” versus “PM”.

Furthermore, the generator was able to create a convincing reply to an email in our custom made corpus. This can be demonstrated through an example. Given the following data from our fictitious victim, Jane Doe:

Abbreviations	“u” for “you”
Capitalization	mixed case
Manually-added signature	“Jane D”
Social groups	`tim@bigcorp.domain`, `rick@bigcorp.domain`
Real names	`rick@bigcorp.domain` ⇒ Rick CoWorker

If Jane Doe had received an email from Tim Boss with the text:

```
Hi Jane,

When you get this message, I'm expecting a reply.
Please reply to me as soon as possible.

Thanks,

T Boss

--
Tim Boss
The Big Manager
Big Corporation
```

And given an intended spam message of:

```
Hi,

I just talked with [INSERT NAME HERE] and you should tak
e a quick look at http://some.bad/url
```

Our generator was able to create a reply to Tim Boss with the text below. (Note that the style of quoted text and the automatically-appended signature were added by the victim's email program.)

```
Tim Boss wrote:
>Hi Jane,
>
>When you get this message, I'm expecting a reply.
>Please reply to me as soon as possible.
>
>Thanks,
>
>T Boss
>
>--
>Tim Boss
>The Big Manager
>Big Corporation
>

Hi,

I just talked with Rick CoWorker and u should take a
quick look at http://some.bad/url
Jane D

--
Jane Doe
An Employee
The Big Corporation
```

Similarly, our generator can forward an existing message from the victim's inbox to a target. Continuing with the previous example, if Jane Doe had an email from Tim Boss in her inbox, our generator could forward this email to Rick CoWorker (part of Jane's social group) and add spam text. This forwarded email might look like:

```
Hi Rick,

I thought u might be interested in knowing about this.
Also, take a quick look at http://some.bad/url for more
details.

-------- Original Message --------

Subject: That message
Date: Fri, 06 Aug 2005 10:01:14 -0600
From: Tim Boss <tim@bigcorp.domain>
To: jane@bigcorp.domain


Hi Jane,

When you get this message, I'm expecting a reply.
Please reply to me as soon as possible.

Thanks,

T Boss

--
Tim Boss
The Big Manager
Big Corporation

--
Jane Doe
An Employee
The Big Corporation
```

Since we had the Enron corpus at our disposal, we analyzed it to determine approximately how many people our spamming technique would have to fool in order to infect any given target within a group of connected people. The Enron corpus contains 517,431 messages from 151 employees' mailboxes. We examined each message, noting who had sent email to whom, using the From:, To:, and CC: fields. This defines a graph where each email address is a node, and each connection (as given by a pair of addresses found in the From-To or From-CC fields) is an edge. Using this graph, we then computed the shortest path between each pair of nodes. (In order to reduce the amount of computation required, we reduced the graph to only consider messages sent between addresses within the `enron.com` domain.) According to our analysis, the maximum path length is 15 hops, with a median of 8 hops. Using this graph we also found that 89 nodes had a degree of 500 or more, which intuitively corresponds to the number of highly connected people in the Enron social group. This "small world" helps support our claim that using a social network is an effective way to distribute more effective spam.

We also analyzed our Enron graph to find pairs of people with close relationships, where two people have exchanged a lot of mail with each other. Such people may fall prey more easily to the special case of forwarded email we mentioned, where the adversary's URL is in the Subject: line and the message body is empty. Looking at both single- and multiple-recipient email messages in the corpus where 500 or more messages have been sent between two people, there are 506 close relationships. If we lower our threshold from 500 to 100 messages, then the number of close relationships climbs to 4511. In context, this means that out of every 46 relationships found in the Enron graph, we find one close relationship. We would conclude that this special case of forwarded email would not yield enough victims by itself for an adversary, yet it may be effective as a complement to other techniques.

New Defenses

User education is often viewed as a key defense against malware and spam. However, this is of limited value against the more effective spam we describe, because the spam bears all the normal hallmarks that the legitimate sender would display. This implies that a strong reliance must be placed on technical defensive measures. We suggest the following:

- Encrypt email corpora. Encryption will delay malware from mining email corpora, but will not indefinitely prevent it. Malware on the zombie machine can eventually steal the decryption key and operate as before.
- Salt email corpora. Email corpora can be salted with false information for malware to find, such as spam trap addresses or manually-added signatures like "Help, my machine is a spam zombie!" This would provide some warning of malware infection.
- Sandbox the browser. URLs followed from email should be viewed in a sandboxed browser, to limit the risk from drive-by downloads. More generally, this precaution applies to any application launched from an email reader.
- Adjust anti-spam filters. Anti-spam filters can be attuned to our more effective spam in four ways:
    1. Watch for known infection indicators, if the adversary's malware uses one.
    2. Place extra emphasis on URLs sent via email. This is unfortunately not always a trivial check, as an adversary may use innocuous-looking jump pages in their spam that redirect to the real web site (Spammer-X, 2004).
    3. Canonicalize incoming email before filtering, undoing known transformations that the adversary's malware performs, before looking for known spam. The obvious disadvantage is the emphasis on known information, leaving lots of leeway for an adversary to slip new spam messages through before defenses have caught up.
    4. Examine alleged reply messages both with and without any quoted text from the original message, to try and expose reply-based spam by stripping off its legitimate cloak.

For threats sent by an adversary as attachments, anti-virus software is still the best defense.

Other Possibilities

There are further possibilities for more effective spam. We have limited ourselves to relatively simple transformations of the adversary's original spam message. We conjecture that to scale the length of the spam (e.g., to the length of a typical 419 scam email) in a manner that would be convincing to a target would require a fairly sophisticated language model. This does not preclude an adversary from doing this, merely makes it more difficult. Furthermore, if excessive time or space were required to do this, malware could transmit necessary information from a victim's corpora to the adversary's computer for offline processing.

Zombies could also be used for targeted marketing by spammers. Currently, spammers can enter keywords describing their target market into programs like Atomic Email Hunter (AtomPark Software, 2005). The program will search for those keywords on a web search engine, then mine the returned web pages for email addresses. These keyword queries could be done just as easily in a victim's email corpora, with the advantage that the social networks that the corpora describe tend to connect people with like interests. Further keyword information is available if the adversary's malware automatically locates and searches a victim's homepage (Culotta, Bekkerman, & McCallum, 2004).

Specific vulnerabilities in mail programs, either mail readers or mail transport programs, could be exploited by mining email corpora too. Malware could be instructed to look for vulnerable versions of software in saved email headers, such as the User-Agent: and Received: lines, in order to identify targets. One defense would simply be to not save these email headers, but this defense would need to be widely-adopted to be effective, and a zombie machine could easily intercept the header information on new incoming email.

One hurdle adversaries face when evading targets' statistical spam filters is that there is no easy way of determining which words are indicative of ham and spam (Graham-Cumming, 2003). A victim's email corpus, however, details the words that have gotten by a target's statistical filter in the past. It also documents a target's own vocabulary, which is likely a strong indicator as to which words a target considers normal, or “hammy”.

Related Work

> ‘Directed attacks can, in theory, work, but require so much work on the spammer's part that the spam-filtering community hasn't seen a true attempt of these in the wild.’ – Zdziarski (2005, p. 137)

The quote above refers to attacks on statistical spam filters, and is also echoed by Graham-Cumming (2003) – the assumption is that an adversary will have an uphill battle, manually guessing at per-user ham words in the absence of feedback from the target. Our work suggests the contrary: that such directed attacks, using a victim's email corpus, are very likely, can be automated, and are scalable.

More effective spam can be both simpler *and* richer than predicted. The number one defense suggested by Graham-Cumming (2003) is to only accept plain-text email, yet our more effective spam will work in both plain-text and HTML formats. Graham (2002) refers to one-line messages with URLs as ‘the spam of the future’, yet we have shown that these emails can be enhanced with mined contextual information in a believable fashion, and this is only the tip of the iceberg.

The closest work to ours is the ‘context-aware’ phishing attacks suggested by Jakobsson (2005). There, a target is tricked into believing a phishing email by exploiting context information, in

conjunction with a well-timed attack. While he gives some specific phishing-related examples, there is no scheme proposed for automating such attacks or generalizing them beyond phishing.

Some facets of our approach have appeared independently, too. Parasitic spam (Swimmer, 2005) would have zombies tag spam onto legitimate outgoing email messages; FitzGerald (2005) mentioned zombies making use of a victim's email program settings.

Conclusion

More effective spam can be sent by using malware on zombie machines to mine data from email corpora. This allows spam to be sent that automatically mimics legitimate email sent by the real owners of the zombie machines, and our proof-of-concept implementation demonstrates that the result can be convincing even to seasoned users. While this more effective spam has not, to our knowledge, been seen in the wild, there are defensive steps that can be taken now to limit its impact when this spam makes its debut.

Acknowledgment

The work of both authors has been supported by grants from the Natural Sciences and Engineering Research Council of Canada. Thanks to the anonymous referees, whose comments were helpful in improving this paper.

References

AtomPark Software (2005). Email extractor [Computer software]. Retrieved 13 January, 2006, from http://www.massmailsoftware.com/extractweb.

CALO Project (2004, March 2). Enron email dataset. Retrieved 1 March, 2006, from http://www.cs.cmu.edu/~enron/enron_mail_030204.tar.gz.

Culotta, A., Bekkerman, R., & McCallum, A. (2004). Extracting social networks and contact information from email and the Web. In First Conference on Email and Anti-Spam.

Delany, M., ed. (2005). Domain-based email authentication using public-keys advertised in the DNS (DomainKeys). Internet draft; work in progress.

Dwork, C., & Naor, M. (1993). Pricing via processing or combatting junk mail. In Proceedings of CRYPTO '92, pages 139–147.

FitzGerald, N. (2005). Why 'user authentication' is a bad idea. In Virus Bulletin Conference, page 226. Abstract only; the comment we refer to was made during his conference presentation.

Gladwell, M. (2002). The Tipping Point: How Little Things Can Make a Big Difference: Little, Brown and Company.

Graham, P. (2002). A plan for spam. Retrieved 7 January, 2006, from http://www.paulgraham.com/spam.html.

Graham-Cumming, J. (2003). Fooling and poisoning adaptive spam filters. Sophos white paper.

Graham-Cumming, J. (2004). How to beat an adaptive spam filter. In MIT Spam Conference.

Jakobsson, M. (2005). Modeling and preventing phishing attacks. In Financial Cryptography '05. Phishing panel.

Levine, J. R. (2005). Experiences with greylisting. In Second Conference on Email and Anti-Spam.

McAfee, Inc. (2005). W32/Mydoom.bb@MM. In McAfee Virus Information Library. Retrieved 2 March, 2006, from http://vil.nai.com/vil/content/v_131856.htm.

Milgram, S. (1967). The small-world problem. Psychology Today, pages 61–67.

Simsek, Ö. & Jensen, D. (2005). Decentralized search in networks using homophily and degree disparity. In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pages 304–310.

Smith, M. A., Ubois, J., & Gross, B. M. (2005). Forward thinking. In Second Conference on Email and Anti-Spam.

Spammer-X (2004). Inside the Spam Cartel: Syngress.

Staniford, S., Paxson, V., & Weaver, N. (2002). How to 0wn the Internet in your spare time. In Proceedings of the 11th USENIX Security Symposium.

Swimmer, D. (2005). The spectre of parasitic spam. Virus Bulletin, pages S2–S4.

Travers, J., & Milgram, S. (1969). An experimental study of the small world problem. Sociometry, 32(4), 425–443.

Tyler, J. R., & Tang, J. C. (2003). When can I expect an email response? A study of rhythms in email usage. In Proceedings of the Eighth European Conference on Computer Supported Cooperative Work, pages 239–258.

Wayner, P. (2002). Disappearing Cryptography: Morgan Kaufmann, second edition.

Williamson, M. M. (2003). Design, implementation and test of an email virus throttle. In 19th Annual Computer Security Applications Conference.

Wong, M., & Schlitt, W. (2005). Sender policy framework (SPF) for authorizing use of domains in E-MAIL, version 1. Internet draft; work in progress.

Zdziarski, J. A. (2005). Ending Spam: No Starch Press.

TTAnalyze: A Tool for Analyzing Malware

Ulrich Bayer & Christopher Kruegel & Engin Kirda
Ikarus Software & Technical University of Vienna

About Author(s)

Ulrich Bayer is a malware researcher at Ikarus Software.
Contact Details: Ikarus Software GembH, Fillgradergasse 7, A-1060 Vienna, Austria
phone +43 1 58995 0, fax +43 1 58995 100, e-mail ub@ikarus.at

Christopher Kruegel is an Assistant Professor at the Technical University of Vienna.
Contact Details: Treitlstrasse 3, Technical University of Vienna, 1040 Austria
phone +43 1 58801 18325, fax +43 1 58801 184391, e-mail chris@seclab.tuwien.ac.at

Engin Kirda is an Assistant Professor at the Technical University of Vienna.
Contact Details: Argentinierstr. 8/184-1, Technical University of Vienna, 1040 Vienna, Austria
phone +43 1 58801 18413, fax +43 1 58801 18491, e-mail ek@seclab.tuwien.ac.at

Keywords

Malware analysis, dynamic analysis, binary analysis.

TTAnalyze: A Tool for Analyzing Malware

Abstract

Malware analysis is the process of determining the purpose and functionality of a given malware sample (such as a virus, worm, or Trojan horse). This process is a necessary step to be able to develop effective detection techniques for malicious code. In addition, it is an important prerequisite for the development of removal tools that can thoroughly delete malware from an infected machine. Traditionally, malware analysis has been a manual process that is tedious and time-intensive. Unfortunately, the number of samples that need to be analyzed by security vendors on a daily basis is constantly increasing. This clearly reveals the need for tools that automate and simplify parts of the analysis process.

In this paper, we present TTAnalyze, a tool for dynamically analyzing the behavior of Windows executables. To this end, the binary is run in an emulated operating system environment and its (security-relevant) actions are monitored. In particular, we record the Windows native system calls and Windows API functions that the program invokes. One important feature of our system is that it does not modify the program that it executes (e.g., through API call hooking or breakpoints), making it more difficult to detect by malicious code. Also, our tool runs binaries in an unmodified Windows environment, which leads to excellent emulation accuracy. These factors make TTAnalyze an ideal tool for quickly getting an understanding of the behavior of an unknown malware.

Introduction

Malware, which is a generic term to denote all kinds of unwanted software (e.g., viruses, worms, or Trojan horses), poses a major security threat to computer users. According to estimates, the financial loss caused by malware has been as high as 14.2 billion US dollars in the year 2005 (Computer Economics, 2006). Unfortunately, the problem of malicious code is likely to grow in the future as malware writing is quickly turning into a profitable business (Symantec, 2005). Malware authors can sell their creations to miscreants, who use the malicious code to compromise large numbers of machines that can then be abused as platforms to launch denial-of-service attacks or as spam relays. Another indication of the significance of the problem is that even people without any special interest in computers are aware of worms such as Nimda or Sasser. This is because security incidents affect millions of users and regularly make the headlines of mainstream news sources.

The most important line of defense against malicious code are virus scanners. These scanners typically rely on a database of descriptions, or signatures, that characterize known malware instances. Whenever an unknown malware sample is found in the wild, it is usually necessary to update the signature database accordingly so that the novel malware piece can be detected by the scan engine. To this end, it is of paramount importance to be able to quickly analyze an unknown malware sample and understand its behavior and effect on the system. In addition, the knowledge about the functionality of malware is important for removal. That is, to be able to cleanly remove a piece of malware from an infected machine, it is usually not enough to delete the binary itself. It is also necessary to remove the residues left behind by the malicious code (such as unwanted registry entries, services, or processes) and undo changes made to legitimate files. All these actions require a detailed understanding of the malicious code and its behavior.

The traditional approach to analyze the behavior of an unknown program is to execute the binary in a restricted environment and observe its actions. The restricted environment is often a debugger, used by a human analyst to step through the code in order to understand its functionality. Unfortunately, anti-virus companies receive up to *several hundred* new malware samples each day. Clearly, the analysis of these malware samples cannot be performed completely manually. Hence, automated solutions are necessary.

One way to automate the analysis process is to execute the binary in a virtual machine or a simulated operating system environment. While the program is running, its interaction with the operating system [1] (e.g., the native system calls or Windows API calls it invokes) can be recorded and later presented to an

[1] Because the vast majority of malware is written for Microsoft Windows, the following discussion considers only this operating system.

analyst. This approach reliefs a human analyst from the tedious task of having to manually go through each single malware sample that is received. Of course, it might still be the case that human analysis is desirable after the automatic process. However, the initial results at least provides details about the program's actions that then help to guide the analyst's search.

Current approaches for automatic analysis suffer from a number of shortcomings. One problem is that malicious code is often equipped with detection routines that check for the presence of a virtual machine or a simulated OS environment. When such an environment is detected, the malware modifies its behavior and the analysis delivers incorrect results. Malware also checks for software (and even hardware) breakpoints to detect if the program is run in a debugger. This requires that the analysis environment is *invisible* to the malicious code. Another problem is when the analysis environment does not monitor the complete interaction with the system. When this happens, the malicious code could evade analysis. This might be possible because there exist thousands of Windows API calls, often with arguments that are composed of complex data structures. Furthermore, the malicious code could also interact directly with the operating system via native system calls. Thus, the analysis environment has to be *comprehensive* and cover all aspects of the interaction of a program with its environment.

In this paper, we describe TTAnalyze, a tool that automates the process of analyzing malware to allow a human analyst to quickly get a basic understanding of the actions of an unknown executable. Running a binary under TTAnalyze results in the generation of a report that contains information to give the human analyst a very good impression about the purpose and the functionality of the analyzed sample. This report includes detailed data about modifications made to the Windows registry and to the file system, information about interactions with the Windows Service Manager and other processes, as well as a complete log of all generated network traffic.

The following list summarizes the key features of TTAnalyze:

- TTAnalyze uses emulation to run the unknown binary together with a complete operating system in software. Thus, the malware is never executed directly on the processor. Unlike solutions that use virtual machines, debuggers, or API function hooking, the presence of TTAnalyze is practically invisible to malicious code.
- The analysis is comprehensive because our system monitors calls to native kernel functions as well as calls to Windows API functions. It also provides support for the analysis of complex function call arguments that contain pointers to other objects.
- TTAnalyze can perform function call injection. Function call injection allows us to alter the execution of the program under analysis and run our code in its context. This ability is required in certain cases to make the analysis more precise.

The remainder of this paper is structured as follows. In Section , we present related work in the field of malware analysis. Section discusses the design and implementation details of our proposed system. Section provides an experimental evaluation of its effectiveness. Finally, Section briefly concludes and outlines future work.

Related Work

Analyzing unknown executables is not a new problem. Consequently, many solutions already exist. These solutions can be divided into two groups: *static analysis* and *dynamic analysis* techniques.

Static analysis is the process of analyzing a program's code without actually executing it. In this process, a binary is usually disassembled first.[2] Then, both control flow and data flow analysis techniques are employed to draw conclusions about the functionality of the program. A number of static binary analysis techniques (Christodorescu & Jha, 2003; Christodorescu, Jha, Seshia, Song, & Bryant, 2005; Kruegel, Robertson, & Vigna, 2004) have been introduced to detect different types of malware. Static analysis has the advantage that it can cover the complete program code and is usually faster than its dynamic counterpart. Its main weakness is that the code analyzed may not necessarily be the code that is actually run. In particular, this is true for self-modifying programs that use polymorphic (Szor, 2005; Yetiser, 1993) and metamorphic (Szor, 2005) techniques and packed executables that unpack themselves during run-time (Oberhumer

[2]Disassembling denotes the process of transforming the binary code into corresponding assembler instructions.

& Molnar, 2004). Also, malicious code can make use of obfuscation techniques (Linn & Debray, 2003) to thwart the disassembly step. The reason is that for certain instruction set architectures (most notably, Intel x86), it is difficult to distinguish between code and data bytes in a file.

Dynamic techniques analyze the code during run-time. While these techniques are non-exhaustive, they have the significant advantage that only those instructions are analyzed that the code actually executes. Thus, dynamic analysis is immune to obfuscation attempts and has no problems with self-modifying programs. When using dynamic analysis techniques, the question arises in which environment the sample should be executed. Of course, running malware directly on the analyst's computer, which is probably connected to the Internet, could be disastrous as the malicious code could easily escape and infect other machines. Furthermore, the use of a dedicated stand-alone machine that is reinstalled after each dynamic test run is not an efficient solution because of the overhead that is involved.

Running the executable in a virtual machine (that is, a virtualized computer) such as one provided by VMware (VMware, 2006) is a popular choice. In this case, the malware can only affect the virtual PC and not the real one. After performing a dynamic analysis run, the infected hard disk image is simply discarded and replaced by a clean one (i.e., so called *snapshots*). Virtualization solutions are sufficiently fast. There is almost no difference to running the executable on the real computer, and restoring a clean image is much faster than installing the operating system on a real machine. Unfortunately, a significant drawback is that the executable to be analyzed may determine that it is running in a virtualized machine and, as a result, modify its behavior. In fact, a number of different mechanisms have been published (Robin & Irvine, 2000; Rutkowska, 2006) that explain how a program can detect if it is run inside a virtual machine. Of course, these mechanisms are also available for use by malware authors.

A PC emulator is a piece of software that emulates a personal computer (PC), including its processor, graphic card, hard disk, and other resources, with the purpose of running an unmodified operating system. It is important to differentiate emulators from virtual machines such as VMware. Like PC emulators, virtualizers can run an unmodified operating system, but they execute a statistically dominant subset of the instructions directly on the real CPU. This is in contrast to PC emulators, which simulate all instructions in software. Because all instructions are emulated in software, the system can appear exactly like a real machine to a program that is executed, yet keep complete control. Thus, it is more difficult for a program to detect that it is executed inside a PC emulator than in a virtualized environment. This is the reason why we decided to implemented TTAnalyze based on a PC emulator.

Note that there is one observable difference between an emulated and a real system, namely speed of execution. This fact could be exploited by malicious code that relies on timing information to detect an emulated environment. While it would be possible for the emulator to provide incorrect clock readings to make the system appear faster for processes that attempt to time execution speed, this issue is currently not addressed by TTAnalyze.

In addition to differentiating the type of environment used for dynamic analysis, one can also distinguish and classify different types of information that can be captured during the analysis process. Many systems focus on the interaction between an application and the operating system and intercept system calls or hook Windows API calls. For example, a set of tools provided by Sysinternals (Russinovich & Cogswell, 2006) allows the analyst to list all running Windows processes (similar to the Windows Task Manager), or to log all Windows registry and file system activity. These tools are implemented as operating system drivers that intercept native Windows system calls. As a result, they are invisible to the application that is being analyzed. They cannot, however, intercept and analyze Windows API calls or other user functions. On the other hand, tools (Hunt & Brubacher, 1999) exist that can intercept arbitrary user functions, including all Windows API calls. This is typically realized by rewriting target function images. The original function is preserved as a subroutine and callable through a trampoline. Unfortunately, the fact that code needs to be modified can be detected by malicious code that implements integrity checking.

TTAnalyze uses a PC emulator and thus has complete control over the sample program. It can intercept and analyze both native Windows operating system calls as well as Windows API calls while being invisible to malicious code. The complete control offered by a PC emulator potentially allows the analysis that is performed to be even more fine-grain. Similar to the functionality typically provided by a debugger, the code under analysis can be stopped at any point during its execution and the process state (i.e., registers and virtual address space) can be examined. Unlike a debugger, however, our system does not have to resort to breakpoints, which are known to cause problems when used for malicious code analysis (Vasudevan & Yerraballi, 2005). The reason is that software breakpoints directly modify the executable and thus, can be

detected by code integrity checks. Also, malicious code was found in the wild that used processor debug registers for its computations, thereby breaking hardware breakpoints.

System Description

TTAnalyze is a tool for analyzing Windows executables (more precisely, files conforming to the portable executable (PE) file format (Microsoft PECOFF, 2006)). To this end, the program under analysis is executed inside a PC emulation environment and relevant Windows API and native system calls are logged. In the following sections, we describe in more detail the design and implementation of key components of TTAnalyze.

Emulation Environment

As mentioned previously, TTAnalyze uses a PC emulator to execute unknown programs. When designing our system, we had to choose between different forms of emulation. In particular, we had to decide if the hardware of a complete PC should be emulated so that an actual off-the-shelf operating system could be installed, or if the processor should be emulated and our own implementation of (a subset of) the operating system interface should be provided. Virus scanners typically emulate the processor and provide a lightweight implementation of the operating system interface (both native system calls and Windows API calls). This approach allows a very efficient analysis process. Unfortunately, it is not trivial to make the operating system stub behave exactly like the actual operating system, and the semantics between a real system and the simulated one differ in many cases. These differences could be detected by malware, or simply break the code. Thus, we decided to emulate an entire PC computer system, running an off-the-shelf Windows XP on top. While the analysis is significantly slower compared to a virus scanner, the accuracy of the emulation is excellent. Since our focus is on the analysis of the behavior of the binary, this trade-off is acceptable.

TTAnalyze uses Qemu (Bellard, 2005), an open-source PC emulator written by Fabrice Bellard, as its emulator component. Qemu is a fast PC emulator that properly handles self-modifying code. To achieve high execution speed, Qemu employs an emulation technique called *dynamic translation*. Dynamic translation works in terms of basic blocks, where a basic block is a sequence of one or more instructions that ends with a jump instruction or an instruction modifying the static CPU state in a way that cannot be deduced at translation time. The idea is to first translate a basic block, then execute it, and finally translate the next basic block (if a translation of this block is not already available). The reason is that it is more efficient to translate several instructions at once rather than only a single one.

Of course, Qemu could not be used in our system without modification. First, it had to be transformed from a stand-alone executable into a Windows shared library (DLL), whose exported functions can be used by TTAnalyze. Second, Qemu's translation process was modified such that a callback routine into our analysis framework is invoked before every basic block that is executed on the virtual processor. This allows us to tightly monitor the process under analysis.

Before a dynamic analysis run is performed, the modified PC emulator boots from a virtual hard disk, which has Windows XP (with Service Pack 2) installed. The lengthy Windows boot-process is avoided by starting Qemu from a snapshot file, which represents the state of the PC system after the operating system has started.

Analysis Process

The analysis process is started by executing the (malware-)program in the emulated Windows environment and monitoring its actions. In particular, the analysis focuses on which operating system services are requested by the binary (i.e., which system calls are invoked). Every action that involves communication with the environment (e.g., accessing the file system, sending a packet over the network, or launching another program) requires a Windows user mode process to make use of an appropriate operating system service. There is no way for a process to directly interact with a physical device, which also includes physical memory. The reason for this stems from the design of modern operating systems, which prohibit direct hardware access so that multiple processes can run concurrently without interfering with each other. Thus, it is reasonable to monitor the system services that a process requests in order to analyze its behavior.

On Microsoft Windows platforms, monitoring system service requests is not entirely straightforward. The reason is that the actual operating system call interface, called native API interface, is mostly undocumented and not meant to be used directly by applications. Instead, applications are supposed to call functions of the documented Windows API. [3] The Windows API is a large collection of user mode library routines, which in turn invoke native API functions when necessary. The idea is that the Windows API adds a layer of indirection to shield applications from changes and subtle complexities in the native API. In particular, the native API may change between different Windows versions and even between different service pack releases. On a Windows system, the native API is provided by the system file `ntdll.dll`. Parts of this interface are documented by Microsoft in the Windows DDK (Microsoft DDK, 2006) and the Windows IFS kit (Microsoft IFS, 2006). Moreover, Gery Nebbett has written an unofficial documentation of the native API (Nebbett, 2000), which covers about 90% of the functions.

Malware authors sometimes use the native API directly to avoid DLL dependencies or to confuse virus scanner's operating system simulations. For this reason, TTAnalyze monitors both the Windows API function calls of an application and also its native API function calls. The task of monitoring which operating system services are invoked by the program requires us to solve two problems:

1. We must be able to precisely track the execution of the malware process and distinguish between instructions executed on behalf of the malware process and those of other processes. This is essential because the virtual processor does not only run the malware process, but also instructions of the Windows operating system and of several Windows' user mode processes. Therefore, a mechanism is required that enables TTAnalyze to determine for each processor instruction whether or not this instruction belongs to the malware process.

2. We need an unobtrusive way for monitoring the accessed operating system services. That is, we have to be able to determine that a native API call or a Windows API call is invoked *without* modifying the malware code. That is, we cannot hook API functions or set debug breakpoints.

We accomplish the precise tracking of the malware process with the help of the `CR3` processor register. The `CR3` register, which is also known as the page-directory base register (PDBR), contains the physical address of the base of the page directory for the current process. The processor uses the page directory when it translates virtual addresses to physical addresses. More precisely, to determine the location of the page directory when performing memory accesses, the processor makes use of the `CR3` register.

Windows assigns each process its own, unique page directory. This protects processes (in particular, their virtual memory address space) from each other by ensuring that each process has its own virtual memory space. The page directory address of the currently running process has to be stored in the `CR3` processor register. Consequently, Windows loads the `CR3` register on every context switch. Thus, we simply have to determine which page directory address has been assigned to the malware process by Windows. Then, we are able to efficiently determine whether or not the current instruction belongs to the test subject under analysis by comparing the current value of the `CR3` register to the page directory address of this test subject.

Determining the physical address of the page directory of the test subject is the responsibility of a probe component that is located *inside* the emulated Windows XP environment. This probe serves as a sensor in the emulated environment and consists of a kernel driver and a program that is run in user mode. The task of the kernel driver is to locate the page directory address that belongs to the test subject and report its findings back to the user mode process. The user mode component then informs TTAnalyze. Note that TTAnalyze is outside the emulated environment, thus, communication between the probe and TTAnalyze has to take place over the virtual network that connects the emulated environment with its host system. To this end, an RPC server is used that runs inside the emulated PC.

The kernel driver is necessary because the page directory address is stored in a memory region that is only accessible to the Windows NT kernel and its device drivers. More precisely, the page directory address can be found as an attribute of that `EPROCESS` structure that corresponds to the test subject. The `EPROCESS` structure is a Windows-internal data object that plays a key role in the way Windows manages processes. For each process in the system, a corresponding `EPROCESS` structure exists. Thus, the device driver has to walk the list of system processes (which consists of `EPROCESS` members) until it finds the one corresponding to the process of the test subject. At this point, the appropriate page directory address

[3] The Windows API is documented by Microsoft in the Platform SDK (Microsoft Platform SDK, 2006).

can be read. Note that the page table address of the test subject's process has to be obtained *before* its first instruction is executed. To this end, the process is created in a suspended state. Only after successfully identifying the page directory address is the test subject allowed to run.

As mentioned previously, the second problem of our analysis is to monitor the invocation of operating system functions.[4] This task can be solved by comparing the current value of the virtual processor's instruction pointer (or program counter) register to the start addresses of all operating system functions that are under surveillance. This comparison is performed in the callback routine of TTAnalyze, which Qemu invokes at the start of each translation block. Note that the start address of a function always corresponds to the first instruction in a translation block. The reason is that a function call is a control transfer instruction, and whenever a control transfer instruction is encountered, Qemu starts a new translation block. At this point, TTAnalyze is invoked and can check the current value of the program counter.

A Windows application typically accesses operating system functions by dynamically linking to system DLLs and calling their exported functions. Thus, we can extract the addresses of interesting functions simply from library export tables. For example, an application calls the Windows API function `CreateFile`, which is implemented in the shared library `Kernel32.dll` when it wants to create a file. In this case, determining the start address of `CreateFile` is easily possible by looking at corresponding entry in `Kernel32.dll`'s export table (and then adding the base address of `Kernel32.dll` to it, as DLLs may be loaded at a different base address).

Function Arguments

Using the system described in the previous sections, we are in a position to know which operating system functions are used by an application. For example, if an application invokes `CreateFile`, we know that a file was created. Unfortunately, we do not dispose of any more details (e.g., the name of the created file). Obviously, we can improve the situation by analyzing the arguments of operating system function calls. To this end, we have extended our analysis framework with the capability to automatically invoke user-specified callback routines in TTAnalyze whenever the test subject calls one of the monitored operating system functions. For each callback routine, the analyst can specify code to process or log the arguments of the corresponding operating system function. For example, if the test subject calls the `CreateFile` function, a TTAnalyze callback routine is invoked where one can access the argument that specifies the name of the file to be created.

To be able to access an argument value of an operating system function, the callback routine has to first read it from the emulated, or virtual, system by specifying its memory address and size. To see this, recall that the TTAnalyze callback routine is running in a different memory address space than the process under analysis. Thus, the writer of a callback routine has to know the size and structure of all function arguments. Reading function call arguments in this fashion would be tedious and error-prone, certainly reducing the number of callback functions. To address this problem, we desire a mechanism to automatically generate the required code for reading the values of functions arguments from the virtual system. The goal is to have the parameter list of a callback routine mirror the parameter list of its corresponding operating system function. Whenever the callback routine is invoked, all function argument values are automatically extracted from the virtual system and then correctly copied into the arguments of the callback routine. In this fashion, the author of a callback function can access the arguments of an operating system function call by simply reading the arguments of the callback routine.

To achieve the goal of generating the necessary C++ source code for reading the arguments of a function call from the virtual system, we developed the *generator* component. This component is a stand-alone program that can be run independently of TTAnalyze. Its task is to generate the desired callback routine stubs (or more precisely, stubs that include the code to handle the arguments). The generator component requires as input a file containing the declarations of all monitored operating system functions. By parsing the function declarations, the generator is able to determine the sizes and structures of functions arguments and can subsequently generate the appropriate C++ code for reading them.

The grammar for the generator's input file resembles the grammar of the C programming language. The difference is that our grammar only supports declarations and no statements. Moreover, we have slightly extended the C-syntax in two ways.

[4] We use the term operating system function as a generic term for both Windows API and native API functions.

1. Parameter declarations of functions may include the keywords `[out]`, `[in]` or `[inout]`. These keywords are used for specifying the direction of a parameter. It effects the point in time when an argument is read. `In` or `inout` parameters are read when a system function call is invoked, while `out` parameters are read when the function returns. If a direction specification is missing, `in` is assumed by default.

2. Array declarations of the form `[ARGx_B]` or `[ARGx_U]` are possible. Such declarations indicate that the variable in front of `[]` is a dynamic array, and that the size of this array is specified by another function argument. The position of this size-specifying argument in the function parameter list is indicated by the value of `x`. Thus, `x` represents an integer value larger than zero. The postfix `_B` further specifies that the size is given in bytes, while the postfix `_U` states that the size is given in units of the array base type. The special form `[NT]` is used for a null-terminated byte array (e.g., C strings are treated as null-terminated byte arrays).

   The reason for having to annotate array arguments is that TTAnalyze has to know how to determine the number of elements of an array during run-time in order to copy the right amount of data to the callback routine. To this end, TTAnalyze can either be told about an argument that specifies the number of array elements, or assume that an array is terminated by a null element. Both cases need to be indicated by proper annotation. As an example, consider the function `int main(int argc, char *argv[])`. This function should be declared as `int main(int argc, char argv[NT][ARG1_U]` in our header file.

For our analysis, we had to manually annotate the function-prototypes in the Windows header files. In particular, we had to assign appropriate qualifiers to output and array parameters.

There is another problem that we have to deal with when reading the values of function arguments from the virtual system. Unfortunately, it is not always immediately possible to read from the virtual address space of a process in the emulated system. To understand this problem, consider that the physical main memory of the emulated PC system simply is a large malloc'ed memory block on the host system. Thus, TTAnalyze can always read from the emulated main memory when supplying a physical address. When supplying a virtual address in the context of the emulated system, however, this virtual address has to be converted into a physical address first. Unfortunately, the possibility exists that the content referred to by this virtual address is not present in the emulated physical memory, but only on the emulated hard disk (i.e., the content is currently paged out). In this case, reading from the virtual system's memory would result in an error. There are also other cases where one is not able to directly retrieve the content for a virtual address. The Windows MMU (memory management unit) uses lazy evaluation as often as possible to save resources (Russinovich & Solomon, 2004). *Lazy evaluation* means to wait to perform a task until it is required. In particular, in the beginning of a process' lifetime, its page tables often do not include shared libraries used by that process. Instead, the page tables are updated only when the processor first references memory in the shared library.

Failing to read an argument of an operating system function call would be a serious drawback. Thus, TTAnalyze must be able to read the memory contents at any specified virtual address. To solve the problem of memory content that is currently paged out, we can resort to the page fault handler of the emulated operating system. More precisely, whenever we wish to access an address that is not present in the emulated physical memory, we force the test subject to read from this virtual address. This read operation invokes the page fault handler of the emulated operating system, which loads the appropriate memory page into the emulated physical memory. When the handler has done its work, the desired content can be easily obtained.

Code Injection

In the previous section, we mentioned the need of TTAnalyze to force the test subject to perform read operations on its behalf. To this end, TTAnalyze has to change the flow of execution of the test subject. This is achieved by *injecting* read instructions into its instruction stream. However, the ability to change the flow of execution of a program is not only useful for inserting read instructions. It can also be used to call arbitrary functions exported by a DLL (e.g., Windows API functions). This ability to insert function calls can be used to improve the quality of the analysis results in the following situations:

- *File created or opened* - The Windows API function `CreateFile` and its native API equivalent `NtCreateFile` can both be used for creating as well as opening a file. There is no way to reli-

ably differentiate between the opening and the creation of a file alone from the arguments used in the function call. To differentiate between these two situations, we have to insert a function that checks whether the file already exists or not. The same situation arises when the Windows API function `RegCreateKeyEx` is called, as `RegCreateKeyEx` can be used for both creating as well as opening a registry key.

- *File or directory* - In several situations, it is not possible to decide if a filename refers to a file or a directory from the function arguments alone.
- *Unknown handles* - TTAnalyze typically monitors all Windows API and native API function calls that return handles. As a result, TTAnalyze knows to what resources these handles refer to. However, handles might be inherited from another process or obtained via a operating system function that is not monitored. In these cases, function call insertion is required to extract information about an otherwise unknown handle.

Because TTAnalyze uses emulation to run the test subject, it is easy to insert additional instructions (such as read instructions) into Qemu's translation blocks. Also, function calls are easy to inject, as a function call is nothing more than a jump to an address (the function start) that is preceded by a push of all function arguments and the return address onto the stack. The main difficulty when performing a function call in the context of the emulated process is that the arguments expected by this function need to be pushed onto the emulated stack. Pushing necessary arguments requires one to serialize and copy all arguments from the host memory into the memory of the emulated system, possibly involving complex function arguments that contain pointers to other structures. This process is the opposite of reading arguments from the emulated system into the host environment. This allows us to reuse the generator component (described in Section) to automatically generate the necessary code to push the arguments onto the emulated stack.

Analysis Report

TTAnalyze is a tool for analyzing malware. While, in principle, arbitrary functions can be monitored, we provide a number of callback routines that analyze and log security-relevant actions. After a run on a test sample, the recorded information is summarized in a concise report. This report contains the following information:

1. General Information - This section contains information about TTAnalyze's invocation, the command line arguments, and some general information about the test subject (e.g., file size, exit code, time to perform analysis, . . .).
2. File Activity - This section covers the file activity of the test subject (i.e., which files were created, modified, . . .).
3. Registry Activity - In this section, all modifications made to the Windows registry and all registry values that have been read by the test subject are described.
4. Service Activity - This section documents all interaction between the test subject and the Windows Service Manager. If the test subject starts or stops a Windows service, for example, this information is listed here.
5. Process Activity - In this section, information about the creation or termination of processes (and threads) as well as interprocess communication can be found.
6. Network Activity - This section provides a link to a log that contains all network traffic sent or received by the test subject.

Evaluation

To demonstrate the capability of TTAnalyze to successfully monitor the actions of malicious code, we ran dynamic tests on current malware samples. Then, we compared the output of our tool to a textual description for each sample. The descriptions that we used were provided by Kaspersky Lab (Kaspersky Lab, n.d.).

The goal of the evaluation was to determine to which extent our analysis results match the characterizations provided by this well-known anti-virus vendor.

For the selection of our test subjects, we consulted Kaspersky's list of the most prevalent malware samples published in December 2005. Unfortunately, it was not possible to obtain samples for all entries on these lists. However, we were able to select ten different malware programs that represent a good mix of different malicious code variants currently popular on the Internet. For some of the names on the list, we received a number of different samples. Some of these samples were packed using different executable packer programs, others were not even recognized as valid Windows PE executables. From this pool, we chose one working sample for each malware type. Then, we scanned all samples for our experiments by the online virus scanner provided by Kaspersky and made sure that they were all recognized correctly.

Malware name	*File*	*Registry*	*Process*	*Service*
`Email-Worm.Win32.Doombot.B`	✗	✗	✗	✗
`Email-Worm.Win32.Netsky.B`	✓	✓	✓	✓
`Email-Worm.Win32.Netsky.D`	✓	✓	✓	✓
`Email-Worm.Win32.Netsky.Q`	✓	✓	✓	✓
`Email-Worm.Win32.Sober.Y`	✓	↯	✓	✓
`Email-Worm.Win32.Zafi.D`	✓	↯	✓	✓
`Net-Worm.Win32.Mytob.BD`	✗	✗	✗	✗
`Net-Worm.Win32.Mytob.BK`	✓	✓	✓	✓
`Net-Worm.Win32.Mytob.C`	✓	↯	✓	✓
`Net-Worm.Win32.Mytob.J`	✗	✗	✗	✗

Table 1: TTAnalyze Test Results

The results of our experiments are shown in Table . In this table, a ✓ symbol indicates that the output of our tool exactly matches the provided description. However, in a surprising number of cases (indicated by the ✗ symbol), the output of our tool differed from the provided description. Interestingly, manual analysis confirmed that our system was indeed producing correct results, and that the behavior provided in the textual description was not reproducible. The differences between the output of our tool and the virus descriptions can have several reasons. In many cases, the general behavior reported by TTAnalyze confirmed the textual description, but the details did not match precisely. For example, both sources reported in agreement that a certain file was created in the system directory, but the file names were different. This can occur when the malicious code chooses random filenames or a name from a list of options that are not exhaustively covered by the malware description. Another reason for differences between our output and a textual description could be that the virus scanner identified an executable as a member of a certain malware variant, while in fact, the behavior of our particular malware instance has slightly changed.

In three cases, which are indicated by the ↯ symbol, our analysis failed to recognize the creation of certain Windows registry values. The reason was that these registry entries were created by the client-server subsystem process `csrss.exe` on behalf of the malicious code. Because our analysis was only recording the actions of the malware itself, we only observed the interaction of the sample with the `csrss.exe` process. However, there is no inherent restriction in TTAnalyze's design that prohibits monitoring more than one process. Thus, by also monitoring the actions of those processes that are interacting with (or started by) the malicious code, such cases can be successfully covered.

Conclusions and Future Work

Because of the window of vulnerability that exists between the appearance of a new malware and the point where an appropriate signature is provided by anti-virus companies, every new malware poses a serious threat to computer systems. This paper introduced TTAnalyze, a system to analyze the behavior of an unknown program by executing the code in an emulated environment. The goal of the analysis process is to gain a quick understanding of the actions performed by malicious code with the general aim of reducing the window of vulnerability. To this end, our tool records the invocation of security-relevant operating system functions (both Windows API functions and native kernel calls).

Because the sample program is executed completely in software on a virtual processor, TTAnalyze can tightly monitor the process without requiring any modifications to its code. This allows the system to easily handle self-modifying code and code integrity checks, two features commonly observed in malware. Furthermore, the emulated system presents itself to running processes exactly like a real system. This makes it more difficult for malware to detect the analysis environment when comparing our solution to virtual machine or debugging environments. Finally, TTAnalyze uses a complete and unmodified version of Windows XP as the underlying operating system in which the unknown program is started. Thus, TTAnalyze provides a perfectly accurate environment for malicious code.

During the course of testing TTAnalyze with real malware samples, it became apparent that dynamic analysis alone is often not sufficient to obtain the complete picture of the behavior of an unknown executable. The reason is that only a single execution path can be examined during a particular analysis run. To address this problem, we aim to extend our analysis so that multiple execution paths can be explored. For example, the process under analysis could be cloned when the emulator encounters a conditional branch. Then, the branch predicate is inverted in one process, causing both processes to follow alternative paths of the program. This could enable us to capture the behavior of an executable in different environments, with different inputs, or under special circumstances (e.g., the executable is run at a certain day of the year such as the now infamous Michelangelo virus that becomes active on the birthday of the famous artist).

References

Bellard, F. (2005). Qemu, a Fast and Portable Dynamic Translator. In *Usenix annual technical conference*.

Christodorescu, M., & Jha, S. (2003). Static Analysis of Executables to Detect Malicious Patterns. In *Usenix security symposium*.

Christodorescu, M., Jha, S., Seshia, S., Song, D., & Bryant, R. (2005). Semantics-Aware Malware Detection. In *Ieee symposium on security and privacy*.

Computer Economics. (2006). *Malware Report 2005: The Impact of Malicious Code Attacks*. `http://www.computereconomics.com/article.cfm?id=1090`.

Hunt, G., & Brubacher, D. (1999). Detours: Binary Interception of Win32 Functions. In *3rd usenix windows nt symposium*.

Kaspersky Lab: Antivirus Software. (n.d.). `http://www.kaspersky.com/`.

Kruegel, C., Robertson, W., & Vigna, G. (2004). Detecting Kernel-Level Rootkits Through Binary Analysis. In *Annual computer security application conference (acsac)*.

Linn, C., & Debray, S. (2003). Obfuscation of Executable Code to Improve Resistance to Static Disassembly. In *Acm conference on computer and communications security (ccs)*.

Microsoft PECOFF. (2006). *Microsoft Portable Executable and Common Object File Format Specification*. `http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.%mspx`.

Microsoft IFS KIT. (2006). `http://www.microsoft.com/whdc/devtools/ifskit`.

Microsoft Platform SDK. (2006). `http://www.microsoft.com/msdownload/platformsdk/`.

Nebbett, G. (2000). *Windows NT/2000 Native API Reference*. New Riders Publishing.

Oberhumer, M., & Molnar, L. (2004). *UPX: Ultimate Packer for eXecutables*. `http://upx.sourceforge.net/`.

Robin, J., & Irvine, C. (2000). Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. In *Usenix annual technical conference*.

Russinovich, M., & Cogswell, B. (2006). *Freeware Sysinternals*. `http://www.sysinternals.com/`.

Russinovich, M., & Solomon, D. (2004). *Microsoft Windows Internals: Windows Server 2003, Windows XP, and Windows 2000*. Microsoft Press.

Rutkowska, J. (2006). *Red Pill... Or How To Detect VMM Using (Almost) One CPU Instruction*. `http://invisiblethings.org/papers/redpill.html`.

Symantec. (2005). *Internet Security Threat Report*. `http://www.symantec.com/enterprise/threatreport/index.jsp`.

Szor, P. (2005). *The Art of Computer Virus Research and Defense*. Addison Wesley.

Vasudevan, A., & Yerraballi, R. (2005). Stealth Breakpoints. In *21st annual computer security applications conference*.

VMware: Server and Desktop Virtualization. (2006). http://www.vmware.com/.

Windows Device Driver Kit 2003. (2006). http://www.microsoft.com/whdc/devtools/ddk/.

Yetiser, T. (1993). *Polymorphic Viruses - Implementation, Detection, and Protection*. http://vx.netlux.org/lib/ayt01.html.

Winning the Battles, Losing the War? Rethinking Methodology for Forensic Computing Research.

Vlasti Broucek and Paul Turner
School of Information Systems, University of Tasmania

About Author(s)

Vlasti Broucek, MSc has been working in the computer industry since 1985, currently, he is a researcher in the School of Information Systems and ICT Manager in the School of Psychology, both at the University of Tasmania, Australia. Vlasti's research focus is on Legal and Technical Issues of forensic computing. Vlasti has an MSc degree from the Czech Technical University in Prague and currently is pursuing a PhD degree at the University of Tasmania under leadership of Dr Paul Turner. Vlasti is publishing extensively in the space of Forensic Computing and received several awards for his work. In May 2004, Vlasti was elected to the post of Scientific Director of European Institute for Computer Anti-virus Research (EICAR).

Mailing address: School of Information Systems, Private Bag 87, Hobart TAS 7001, Australia; Phone: +61-3-62262346; Fax: +61-3-62262883; E-mail: Vlasti.Broucek@utas.edu.au

Associate Professor Paul Turner is a Senior Research Fellow at the School of Information Systems, University of Tasmania where he is the University's research coordinator for the Smart Internet CRC (http://www.smartinternet.com.au). Paul also currently leads a group of five researchers in the e-forensics and computer security domains. Prior to joining the University in 2000, Paul was a research fellow at CRID (Computer, Telecommunications and Law Research Institute) in Belgium. Paul has also worked as an independent ICT consultant in Europe and was for 3 years editor of the London-based Telecommunications Regulation Review.

Mailing address: School of Information Systems, Private Bag 87, Hobart TAS 7001, Australia; Phone: +61-3-62266240; Fax: +61-3-62266211; E-mail: Paul.Turner@utas.edu.au

Keywords

Forensic Computing, Cyber Crime, Methodology, Privacy, Security.

Winning the Battles, Losing the War? Rethinking Methodology for Forensic Computing Research.

Abstract

In the last ten years Forensic Computing (FC) has emerged in response to the challenges of illegal, criminal and other inappropriate on-line behaviours. As awareness of the need for the accurate and legally admissible collection, collation, analysis and presentation of digital data has grown, so has recognition of the challenges this requirement poses for technical, legal and organisational responses to these on-line behaviours. Despite recognition of the multi-dimensional nature of these behaviours and the challenges faced, agreement on coherent frameworks for understanding and responding to these issues, their impacts and their interrelationships appears to remain a long way off. As a consequence, while significant advances have been made within technical, organisational and legal 'solution centred paradigms', the net result appears to be a case of 'winning the battles but losing the war' on computer misuse and e-crime.

This paper examines this situation and reflects on its implications for academic researchers' methodological approach to understanding and responding to these challenges. This paper suggests the need to reconceptualise the term 'solution' and advocates an additional methodological step, (that it is anticipated will generate data) for the development of a framework to map the value propositions of, and interrelationships between the individual sets of responses within the dynamically evolving forensic computing landscape. By exposing issues, responses and underlying assumptions it is anticipated that this will improve the possibility of calibrated responses that more effectively and coherently balance the interests for security, privacy and legal admissibility.

Introduction

The last decade has seen an explosion of research and development by computer security specialists, legal professionals, information managers and others on technical, legal and organisational issues arising from illegal, criminal and/or inappropriate on-line behaviours. Researchers drawn from a variety of disciplines have explored different aspects of the issues and advocated different solutions to these forensic computing challenges (for example, Reno, 1996; Venema & Farmer, 1995; Verreck, 2000a, 2000b). To date however, despite numerous attempts there is a lack of agreement on frameworks for either understanding and/or responding to these issues, their impacts and their interrelationships. This can partly be explained by the fact that with growing demand for practical solutions to the challenges faced, different business, legal and organisational imperatives drive developments in ways that militate against coherence in the name of competitive advantage.

Significantly however, it is evident that this lack of coherence is of more than purely academic interest and has directly inhibited awareness of the issues and challenges in the community, impaired the development and diffusion of specialised forensic computing skills and, most importantly, impacted directly on the effectiveness of responses to computer misuse and e-crime (Broucek & Turner, 2005; Broucek, Turner, & Frings, 2005). Indeed, whilst many forensic computing specialists have advocated the need for more integrated solutions that balance the requirements for network security, individual privacy and legally admissible digital evidence, there remains little evidence of these emerging.

From the perspective of academic research this situation raises a series of questions about the nature of the discrete technical, organisational and socio-legal responses being developed towards computer misuse and e-crime, including:

- What is the nature of the frameworks that individual sets of responses use to conceptualise the issues and challenges their solutions aim to address?
- How do individual sets of responses conceptualise the relationships between themselves and other sets of responses?
- What key value propositions underpin individual sets of responses and how does this influence their perceptions of what constitutes a solution?
- How can a framework be developed that will more explicitly map the value propositions of, and interrelationships between the individual sets of responses?

This paper considers these issues and reflects on their implications for academic researchers' methodological approach to forensic computing research. The paper commences with a review of attempts to define and model the forensic computing domain. This review highlights the limited agreement that exists on models for understanding and/or responding to issues, their impacts and interrelationships. The paper also presents evidence highlighting how the interrelatedness of these issues means that responses in one area can have negative consequences for developments in another area, thereby inhibiting the overall effectiveness of the current approaches. On the basis of this analysis, the paper acknowledges that whilst, in the short term at least, technical, organisational and legal responses to computer misuse and e-crime will remain fragmented, academic researchers need to reconceptualise the notion of a 'solution' and take steps to develop a framework to map the value propositions of, and interrelationships between these individual sets of responses. In this context, the paper outlines an additional methodological step that is anticipated to generate data that will enable the development of this framework. By exposing issues, responses and underlying assumptions it is anticipated that this will improve the possibility of calibrated responses that more effectively and coherently balance the interests for security, privacy and legal admissibility.

Models and Frameworks: A Help or Hindrance?

Previous research has identified the absence of an overarching conceptual framework for forensic computing and revealed how this has contributed to limiting exploration of the interdisciplinary dimensions of issues concerned with the identification, collection and analysis of computer evidence (Broucek & Turner, 2001a, 2001b). However, given that this paper is directly concerned with the reasons for, and methodological implications of this lack of coherence, the first part of this paper reviews existing models and frameworks as a way to explore their differences and underlying suppositions.

In this context, it is first important to examine the definitional ambiguity that has surrounded forensic computing itself. In this regard the following definitions can be presented as being representative of the range of definitional approaches:

- 'The process of identifying, preserving, analysing and presenting digital evidence in a manner that is legally acceptable'. (McKemmish, 1999)
- 'Gathering and analysing data in a manner as free from distortion or bias as possible to reconstruct data or what has happened in the past on a system'. (Farmer & Venema, 1999)
- 'The use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation and presentation of digital evidence derived from digital sources for the purpose of facilitating or furthering the

reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations.' (Palmer, 2001)

As the above quotes indicate, Forensic[1] Computing has continued to remain problematic to define. Broucek & Turner's (2001a; , 2001b) preliminary taxonomy of forensic computing (FC) highlights how, as an academic discipline, FC builds on knowledge drawn from several other fields of expertise. This taxonomy also illustrates the broad range of issues and approaches within FC and suggests the following working definition of FC as being:

- 'Processes or procedures involving monitoring, collection, analysis and presentation of digital evidence as part of "a priori" and/or "post-mortem" investigations of criminal, illegal or other inappropriate on-line behaviours.'

Subsequent work expanded this initial taxonomy to include law enforcement and the basic taxonomy of the FC domain is illustrated in Figure 1. For further information on this approach refer to (Broucek & Turner, 2001a, 2001b; Hannan, Frings, Broucek, & Turner, 2003; Hannan, Turner, & Broucek, 2003). It should however be noted that this taxonomy is by no means the only attempt to define the domain.

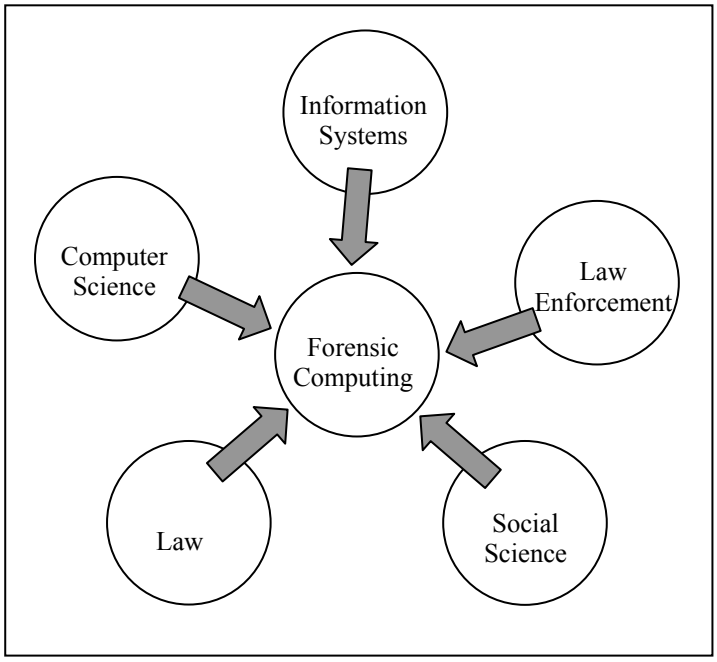


Figure 1: Forensic Computing Domain

Indeed, as a result of work at the First Digital Forensic Workshop (DFRWS) in 2001 an entirely different approach based not on constitute disciplines but rather specific fields of computer forensic activity led to the development of an alternative model referred to as the "Nucleus of Digital Forensic Research". This model is displayed in Figure 2 below.

[1] It is useful to note that the term 'forensic' is defined as: 'used in or connected with court of law' (Hanks, 1991)

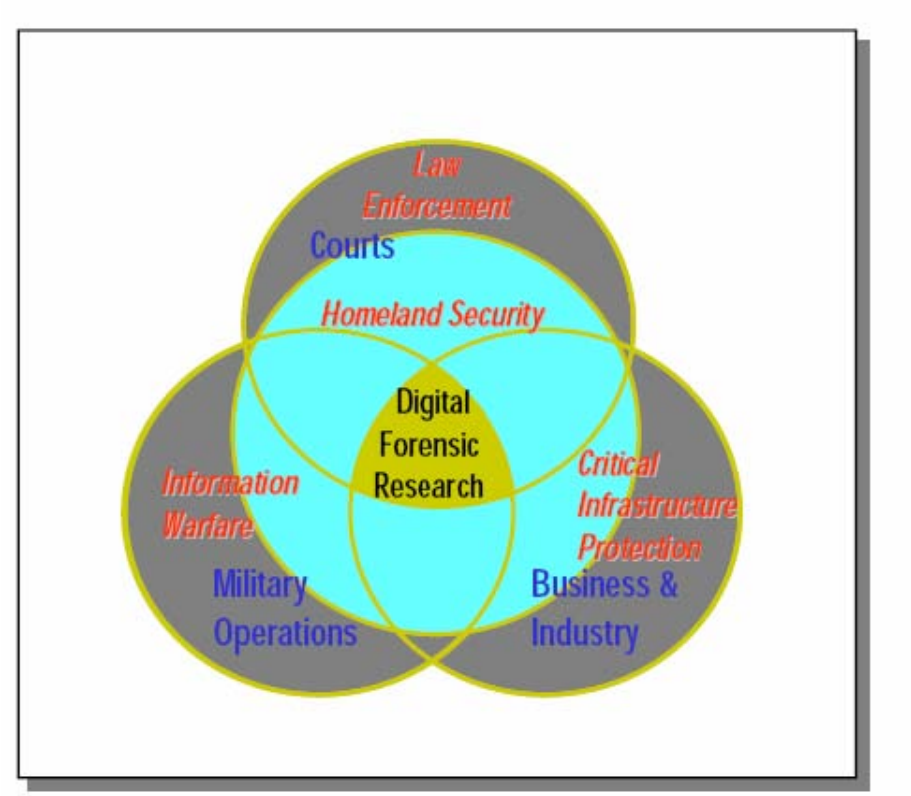


Figure 2: Nucleus of Digital Forensic Research (from Palmer, 2001)

Interestingly both the 'taxonomy' and 'nucleus' frameworks emphasize the need to stimulate cooperation and collaboration amongst the various disciplines and fields concerned with forensic computing issues. The same urgency to support interaction and collaboration has also been suggested by Spafford (cited in Palmer, 2001, p. 7). Spafford argues that it is necessary

- To abandon current "band aid approach" to forensics. The same approach was and still is often observed in the security world – the security elements are added into the existing or new systems, instead of designing the systems with security already built in;
- To know exactly how much information and what type of it needs to be collected for further analysis in particular circumstances;
- To understand social aspects of 'the game'.

Spafford concludes that in the forensic computing domain 'All aspects of the problem are essential. Therefore, it is imperative that each collaborates with the other. Researchers, investigators, legislators, and jurists must all work toward a central goal. This requires constant discussion within groups that have representation from all essential parties' (Palmer, 2001, p. 8).

What is significant about these remarks is that despite there apparent conformity and agreement, an academic analysis of the work arising from, for example, the First Digital Forensic Research Workshop, reveals the markedly different aims/goals/objectives underpinning the approaches of different domain experts.

As the presentations of the main speakers at the DFRWS workshop illustrate (i.e. Eugene Spafford representing academic research and government, Charles Boeckman representing DOD Operations, Chet Hosmer representing Commercial Tools development, David Baker representing Critical

Infrastructure Protection and John Hoyt representing Law Enforcement), the different underlying positions can be summarised as follows:

- Law enforcement agencies appear primarily interested in gathering evidence that can later be used for prosecution[2];
- Business requirements are more or less driven by economic pressures of remaining viable and competitive;
- Academics are interested in exact, scientific methods and data and the advancement of new knowledge;
- Military and Information Warfare Operations are mainly interested in what is referred to as Defensive Information Operations (DIO). DIO represents a multi-disciplinary approach to protecting digital systems. It includes Communication, Computer, Information and Operational Security as well as Physical Security, and other tactics used in active systems protection[3].

The result of DIO is research conducted in defence operations concentrated mainly on:

- Optimisation of data collection,
- Minimisation of data corruption or destruction risks, and
- Accommodation of operational time constraints.

Table 1, adapted and extended from Palmer (2001), clearly demonstrates that investigators from each area deploy different paradigm when approaching FC and its analysis. Furthermore, the workshop appears to agree that they also attempt to do this in different environments (that the authors here would prefer to call time frames). Significantly, it is suggested that Law Enforcement is only interested in "post-mortem" while all the other players tend to anticipate and try to take an action to thwart a possible threat even before it happens including such factors as financial cost, reputation, service availability.

Area	**Primary Objective**	**Secondary Objective**	**Environment - Time Frames**
Law Enforcement	Prosecution		After the act/post-mortem
Military IW Operations	Continuity of Operations	Prosecution	Real Time
Business and Industry	Availability of Services	Prosecution	Real Time
Academics	Advancement of knowledge	Dissemination of knowledge	Variable: subject to externalities

Table 1: Suitability Guidelines for Digital Forensic Research (adapted and extended based on Palmer (2001).

It is noticeable that the approach advocated by DFRWS can be directly linked to models that subsequently emerged focused on how to respond in conducting/implementing FC investigations. Broadly these models can be divided into three categories Simple, Advanced and Complex. These are discussed below.

[2] It is worth pointing out that such evidence must follow rules of evidence established by particular jurisdiction to maintain evidential integrity (chain of custody) and that the digital evidence should form a part of 'whole case' and non-technical elements should also be taken into an account.

[3] The main differentiation of defence operation requirements from those in law enforcement is willingness of DIO to sacrifice absolute and/or even measurable accuracy for quickness in order to serve a mission's timeline.

Simple Models

Arising directly out of the DFRWS was the development of 7 Step linear model for the conduct of FC investigations:

- Identification
- Preservation
- Collection
- Examination
- Analysis
- Presentation
- Decision

Reith, Carr, and Gunsch (2002) extended this model to nine steps and called it the Abstract Digital Forensic Model. The nine steps included are:

- Identification
- Preparation
- Approach strategy
- Preservation
- Collection
- Examination
- Analysis
- Presentation
- Returning Evidence

These models concentrate on processing digital evidence. They do not identify flow in investigation, do not include issues like chain of custody and different requirements and needs of different groups of users as defined above and in (Palmer, 2001).

Advanced Model

Carrier & Spafford (2003) after analysis of several other models developed their 'Integrated Digital Investigation Process (IDIP)'. This model is based on crime scene theory for physical investigation. They argue that this investigation process has been refined over time through its use in thousands of investigations, and as such, is the most suitable model upon which to establish a model for digital investigations. As a result their model is premised on the assumption that the computer should be treated as a separate crime scene to the extent that they use the analogy that the computer should be treated the same as a "*body at the murder scene*". (Carrier & Spafford, 2003, p. 18). Carrier & Spafford's model has been extended more recently by Baryamureeba & Tushabe (2004) with their Enhanced Digital Investigation Model (EDIP). The EDIP expands the IDIP to enable tracing back to address the issues of digital investigations in networked and wireless world. However, even with this enhanced model it is useful to consider how far the physical investigation analogy can be pushed given the challenges of incidental/multiple digital copies of any individual data element and the capacity of systems to 'tamper' with data during analysis.

Complex models

Most recently, one of the most complex models/frameworks has been developed by CTOSE. Discussion of this model has been widely published and is can be examined in depth through the following papers (Broucek & Turner, 2004a; Broucek, Turner, & Frings, 2005; CTOSE, 2003;

Leroux & Pérez Asinari, 2003; Sato, Broucek, & Turner, 2005; Urry & Mitchison, 2003) and at CTOSE website (http://www.ctose.org/). In summary, the CTOSE model was developed as a high level tool aimed to assist companies or other individuals/groups to respond correctly during the investigation and analysis of on-line behaviours in order to generate legally admissible evidence. The CTOSE approach was to develop an 'expert system' to guide users (for example, a company's security officer) in their preparations and responses to e-security incidents.

Importantly, the CTOSE approach aim is to bring to these organisations an overall benchmark against which to compare their own operations and procedures relating to evidence handling. CTOSE framework offers integrated functionality across whole spectrum of involved actors in different groups that have a stake in the process of evidence handling, whether organisations, IT, law enforcement or the legal establishment. It appears that CTOSE framework is the only framework to be comprehensive across the entire chain of evidence handling, flexible across different types of organisations as well as portable across countries. However, questions now arise over adoption and use of this comprehensive approach and what factors would stimulate more widespread implementation of the approach.

Finally, it is useful to review(Ciardhuáin, 2004) 13 step model that integrates each step with sequence of activities and information flows. To date, attempts have been made to validate this model through the conduct of interviews with police investigators and to deploy the model as part of genuine police computer forensic investigations. The thirteen steps of this model are:

- Awareness
- Authorisation
- Planning
- Notification
- Search for and identify evidence
- Collection of evidence
- Transport of evidence
- Storage of evidence
- Examination of Evidence
- Hypothesis
- Presentation of hypothesis
- Proof/Defence of hypothesis
- Dissemination of information

In summary, it is evident that despite numerous attempts to model the forensic computing domain definitional heterogeneity remains. It is evident that this heterogeneity is at least partly due to the differing assumptions, aims, goals and objectives underpinning the approaches outlined above. Significantly, however despite widespread recognition of the multi-dimensional nature of on-line behaviours and the challenges of understanding and responding to them, little progress has been made. Indeed, even where comprehensive frameworks have been developed the additional challenges of adoption and utilisation have emerged to limit the benefits as the problems of computer misuse and e-crime continue to grow.

While some readers may consider these issues merely of academic interest, the next section highlights how this lack of coherence has directly inhibited awareness of the issues/challenges in the community, impaired the development and diffusion of specialised forensic computing skills and, most importantly, impacted directly on the effectiveness of responses to computer misuse and e-crime (Broucek & Turner, 2005; Broucek, Turner, & Frings, 2005).

Winning the Battle and Losing the War

The increasing incidence of computer misuse and e-crime has led to strong demand across the public and private sectors for effective ways to address these behaviours and has contributed to the stimulation of research, development and commercialisation of technical, organisational and socio-legal responses. Whilst individual responses can be complemented for their innovation in continuing to address the evolving challenges faced, it has become increasingly obvious that truly effective offensive and defensive solutions will require both integration and implementation of insights from each.

Following Spafford (Palmer, 2001, p. 7) further research must address individual challenges in the technical, procedural, social, and legal realms as well as the integration between them 'if we hope to craft solutions that begin to fully "heal" rather than constantly "treat" our digital ills'. More specifically, Spafford advocates the need to 'incorporate forensic hooks into tools rather than use our current band aid approach that produces point solution tools' and also mechanisms to begin to answer the problem of training and experience. Much more effort is required in producing user interfaces that address deficiencies in skill levels that will always be with us and will no doubt get worse as the problems grow. We need to know how much information and what type exactly we must collect to afford the most accurate analysis under particular circumstances. Common terms of reference are needed as well as common analytical standards and practices'.

In working towards more integrated solutions that balances requirements for network security, individual privacy and the need for legally admissible digital evidence it is useful to reiterate the recommendations previously articulated by Broucek & Turner (2004a)

- Best practice for digital evidence handling should involve deployment of the highest investigative standards at all stages in the identification, analysis and presentation of digital data;
- Targeted training and education of network administrators and end-users in the key principles of digital evidence handling is urgently required as well as education and awareness amongst users of the consequences of their on-line behaviours for system security;
- Opportunities exist for the further refinement of e-forensic methodologies and processes such as those developed by CTOSE and these must include a recognition of the dynamic and multi-faceted nature of the forensic computing domain;
- Enhancing e-forensic professionalism through the rapid development of processes for e-forensic computing competences and certification is an essential element in building and implementing integrated solutions[4].

Unfortunately, despite these recommendations and recognition of the need for integrated solutions there is still only limited evidence that these are actually emerging. While the lack of collaboration and integration can be at least partially explained by the complexity of the individual sets of issues faced, it is clear that this fragmentation of effort is inhibiting the development of awareness and specialised skills required as well as having detrimental impacts on individual responses developed. Following Broucek & Turner's (2005) argument, there is the critical need for the development of integrated solutions that acknowledge how in digital environments developments in one area have serious implications for developments in another. Their paper revealed how without a conscious recognition of the interrelatedness of these responses we will continue to create vulnerabilities

[4] Interestingly, a degree in computer science is till sufficient qualification for expert witness in court proceedings

and/or problems that may actually impair the effectiveness of our overall response to computer misuse and e-crime – this in turn impacts directly on the ability of industry, government or academia to improve things i.e. we are 'winning our respective battles but possibly losing the war'.

The commercial pressures driving research, development and commercialisation further complicate the situation. These pressures often militate against mutual cooperation. For example, major players in the Computer Anti-virus industry still cannot reach agreement on a common naming system. This appears to be partly because of different organisations desire to acquire a "commercial edge". Another example of development of security tools without taking in account forensic capabilities has been identified by the authors' recent analysis of a premier digital data visualisation tool that appears to have been constructed and developed without consideration as to whether or not the output of its analysis would be legally admissible.

It is also evident that the marketplace sees commercial benefit from developing tools with anti-forensic capabilities. While it is acknowledged that currently there is not widespread awareness of these tools, perhaps this is only a matter of time. It can also be anticipated that cyber-criminals are already aware of and using these commercially available tools. For example, tools like CIPHER.EXE included in the standard distribution of WindowsXP can significantly hinder forensic analysis. While some recent research (for example, Mathew Geiger's work at Carnegie Melon University) suggests that many of these anti-forensic tools are less effective than they claim to be, their existence is illustrative of the nature of the problems faced.

While perhaps, for the near future at least, we may have to accept continued fragmentation of efforts there remains a clear need for consideration of how well each approach balances the interests of security, legal admissibility and privacy should be incorporated into our discussions. Ultimately of course we also need to remember that the digital domain itself is also intimately related to the physical world where corroborative evidence and conventional investigative techniques have a role to play (Broucek & Turner, 2005). For academic researchers this situation also presents a challenge in terms of how to conduct on-going research into forensic computing and suggests the need for researchers to reconceptualise the notion of a 'solution' and take steps to develop a framework to map the value propositions of, and interrelationships between these individual sets of responses. The next section of the paper examines this approach.

Methodological Implications for Forensic Computing Research

In the context of the discussion above this section reflects on the implications of the current landscape for academic researchers' methodological approach to forensic computing research.

Previous work by the authors has:

- Developed a taxonomy of forensic computing and explored models and frameworks for understanding and responding to the issues, their impacts and interrelationships arising from online behaviours (Broucek & Turner, 2001a, 2001b; Hannan, Frings, Broucek, & Turner, 2003; Hannan, Turner, & Broucek, 2003)
- Investigated and analysed technical, organisational and socio-legal approaches and consequences (Broucek, Frings, & Turner, 2003; Broucek & Turner, 2003b, 2003c, 2004a, 2004b; Broucek, Turner, & Frings, 2005; Sato, Broucek, & Turner, 2005)
- Advocated approaches to education and training as well as the conduct of forensic computing investigations(Broucek & Turner, 2002a, 2002b, 2002c, 2003a)
- Explored implications of uncoordinated approaches of ensuring more integrated solutions(Broucek & Turner, 2002a, 2005)

However, as the discussion above illustrates an additional methodological step is required to generate data that will reveal the value propositions, attitudes, insights and experiences of domain experts in each stream of research, development and commercialisation. It is anticipated that by generating this data it will be possible to develop a framework that will more clearly expose the issues, responses and underlying assumptions and thereby contribute to improving the possibility of calibrated responses that more effectively and coherently balance the interests for security, privacy and legal admissibility.

At the broadest level, it is argued that the data required can be generated through the conduct of qualitative semi-structured interviews selected domain experts using methods of grounded theory for analysis of the data. More specifically, this section argues the need to explore a series of research questions about the nature of the discrete technical, organisational and socio-legal responses being developed towards computer misuse and e-crime including (but not limited to):

- What is the nature of the frameworks that individual sets of responses use to conceptualise the issues and challenges their solutions aim to address?
- How do individual sets of responses conceptualise the relationships between themselves and other sets of responses?
- What key value propositions underpin individual sets of responses and how does this influence their perceptions of what constitutes a solution?
- How can a framework be developed that will more explicitly map the value propositions of, and interrelationships between the individual sets of responses?

Interview Process

The interviews will be conducted using a semi-structured question frame to guide the flow of interview. Transcripts will be constructed and analysed via a coding process drawing on the principles of grounded theory (Glaser & Strauss, 1967) (Glaser & Strauss, 1967; Strauss & Corbin, 1998). The question frame will be divided into the following sections:

- Section 1: Participant's Background
  The aim of the first section of questions is to collect background information about the participant, his/her knowledge of and involvement in Forensic Computing and his/her organisation. Questions will be framed to determine core business of the organisation and its requirement for Forensic Computing readiness.
- Section 2: Forensic Computing Expectations
  The questions in the second section will focus specifically on expectations that participant from different backgrounds and areas (research/law enforcement/defence/government/ISP) have from Forensic Computing. It is expected that the expectations of each group may be completely different as suggested in the literature review part of the research.
- Section 3: Forensic Computing Problems and Issues
  Section three questions will ask to explain the problems and issues that participants face in day to day life. The participants will be guided towards various technical, legal, organisational and other issues identified in this research as well as in previous section of the interview.
- Section 4: Solutions to Identified Problems
  Section four questions will be aimed at identifying possible solutions and/or at least improvements for the issues identified in section three of the interview. Specific questions will be given to identify critical opinions about who, when and how should provide solutions/tools/answers for the issues identified.

The aim of the questions will be to encourage participants to discuss issues related to the study without imposing limitations or constraints on how the participants answer these questions (Doolin, 1996). The main target is to explore with the experts in the spaces/paradigms:

- Their attitudes, experiences, insights in relation to their domain expertise and how they view the e-forensics space
- The assumptions underpinning the way they navigate, move forward in their domains and how they identify problems, generate responses and evaluate what they do – their 'solutions'
- Their perception of causes for the criminal on-line behaviours and other incidents involving forensic computing
- Recommendations for moving forward

Conclusion

This paper has identified a series of issues addressing the realities of research work in the forensic computing domain. From an academic perspective it has reviewed current understandings and issues arising due to the lack of coherent approaches. It has also identified the implications of current situation for methodologies used by academic researchers and suggested an additional step that may generate data that can be used to enhance the coherency of responses being developed.

This paper strongly demonstrates the need for developing a procedure to understand and model competing requirements for digital data investigations (forensic computing). It proposes to use semi-structured interviews with experts from various organisations followed by grounded theory analysis of the interviews to:

- Better understand the needs of various groups,
- Find commonalities between these groups, and
- Help researchers in the field of Information Systems to better understand needs and requirements for further research in this ever evolving field.

This paper is the first step in generating the necessary data and the authors look forward to implementing aspects of this methodological approach through frank and vigorous interaction with experts present at this year's EICAR conference. The authors anticipate that this additional data collection and analysis will aid in the development of a framework to map the value propositions of, and interrelationships between the individual sets of responses within the dynamically evolving forensic computing landscape. By exposing issues, responses, underlying assumptions and causalities it is anticipated that this will improve the possibility of calibrated responses that more effectively and coherently balance the interests for security, privacy and legal admissibility.

References

Baryamureeba, V., & Tushabe, F. (2004). The Enhanced Digital Investigation Process Model: Makerere University Institute of Computer Science, Uganda.

Broucek, V., Frings, S., & Turner, P. (2003). The Federal Court, the Music Industry and the Universities: Lessons for Forensic Computing Specialists. In C. Valli & M. Warren (Eds.), *1st Australian Computer, Network & Information Forensics Conference*. Perth, WA, Australia.

Broucek, V., & Turner, P. (2001a). Forensic Computing: Developing a Conceptual Approach for an Emerging Academic Discipline. In H. Armstrong (Ed.), *5th Australian Security Research Symposium* (pp. 55-68). Perth, Australia: School of Computer and Information Sciences, Faculty of Communications, Health and Science, Edith Cowan University, Western Australia.

Broucek, V., & Turner, P. (2001b). Forensic Computing: Developing a Conceptual Approach in the Era of Information Warfare. *Journal of Information Warfare, 1*(2), 95-108.

Broucek, V., & Turner, P. (2002a). Bridging the Divide: Rising Awareness of Forensic Issues amongst Systems Administrators. In *3rd International System Administration and Networking Conference*. Maastricht, The Netherlands.

Broucek, V., & Turner, P. (2002b). E-mail and WWW browsers: A Forensic Computing perspective on the need for improved user education for information systems security management. In M. Khosrow-Pour (Ed.), *2002 Information Resources Management Association International Conference* (pp. 931-932). Seattle Washington, USA: IDEA Group.

Broucek, V., & Turner, P. (2002c). Risks and Solutions to problems arising from illegal or Inappropriate On-line Behaviours: Two Core Debates within Forensic Computing. In U. E. Gattiker (Ed.), *EICAR Conference Best Paper Proceedings* (pp. 206-219). Berlin, Germany: EICAR.

Broucek, V., & Turner, P. (2003a). A Forensic Computing perspective on the need for improved user education for information systems security management. In R. Azari (Ed.), *Current Security Management & Ethical Issues of Information Technology*. Hershey, PA 17033-1117, USA: IGP/INFOSCI/IRM Press.

Broucek, V., & Turner, P. (2003b, May 8-9). *Intrusion Detection Systems: Issues and Challenges in Evidence Acquisition.* Paper presented at the CTOSE Conference, Facultés Universitaires Notre-Dame De la Paix, Namur, Belgium.

Broucek, V., & Turner, P. (2003c). Intrusion Detection: Forensic Computing Insights arising from a Case Study on SNORT. In U. E. Gattiker (Ed.), *EICAR Conference Best Paper Proceedings*. Copenhagen, Denmark: EICAR.

Broucek, V., & Turner, P. (2004a). Computer Incident Investigations: e-forensic Insights on Evidence Acquisition. In U. E. Gattiker (Ed.), *EICAR Conference Best Paper Proceedings*. Luxembourg, Grand Duchy of Luxembourg: EICAR.

Broucek, V., & Turner, P. (2004b). Intrusion Detection: Issues and Challenges in Evidence Acquisition. *International Review of Law, Computers and Technology, 18*(2), 149-164.

Broucek, V., & Turner, P. (2005). "Riding Furiously in All Directions" - Implications of Uncoordinated Technical, Organisational and Legal Responses to Illegal or Inappropriate On-line Behaviours. In P. Turner & V. Broucek (Eds.), *EICAR 2005 Conference Best Paper Proceedings* (pp. 190-203). Saint Julians, Malta: EICAR.

Broucek, V., Turner, P., & Frings, S. (2005). Music piracy, universities and the Australian Federal Court: Issues for forensic computing specialists. *Computer Law & Security Report, 21*(1), 30-37.

Carrier, B. D., & Spafford, E. H. (2003). Getting Physical with the Digital Investigation Process. *International Journal of Digital Evidence, 2*(2).

Ciardhuáin, S. Ó. (2004). An Extended Model of Cybercrime Investigation. *International Journal of Digital Evidence, 3*(1).

CTOSE. (2003). *CTOSE Project Final Results*.

Doolin, B. (1996). Alternative views of case research in information systems. *Australian Journal of Information Systems, 3*(2), 21-29.

Farmer, D., & Venema, W. (1999). Murder on the Internet Express. Retrieved November 6, 2001, from http://www.fish.com/forensics/class.html

Glaser, B. G., & Strauss, A. (1967). *The discovery of grounded theory: strategies for qualitative research* Chicago: Aldine Pub. Co.

Hanks, P. (Ed.). (1991). *The Collins Australian pocket Dictionary of the English Language*: HarperCollins Publishers.

Hannan, M., Frings, S., Broucek, V., & Turner, P. (2003). Forensic Computing Theory & Practice: Towards developing a methodology for a standardised approach to Computer misuse. In S.-A. Kinght (Ed.), *1st Australian Computer, Network & Information Forensics Conference*. Perth, WA, Australia.

Hannan, M., Turner, P., & Broucek, V. (2003). Refining the Taxonomy of Forensic Computing in the Era of E-crime: Insights from a Survey of Australian Forensic Computing Investigation (FCI) Teams. In *4th Australian Information Warfare and IT Security Conference* (pp. 151-158). Adelaide, SA, Australia.

Leroux, O., & Pérez Asinari, M. V. (2003, May 8-9). *Collecting and Producing Electronic Evidence in Cybercrime Cases.* Paper presented at the CTOSE Conference, Facultés Universitaires Notre-Dame De la Paix, Namur, Belgium.

McKemmish, R. (1999). What is Forensic Computing. *Trends and Issues in Crime and Criminal Justice*(118).

Palmer, G. (2001). *A Road Map for Digital Forensic Research: Report From the First Digital Forensic Research Workshop (DFRWS)* (DFRWS Technical Report No. DTR-T001-01 Final). Utica, New York: .

Reith, M., Carr, C., & Gunsch, G. (2002). An Examination of Digital Forensic Models. *International Journal of Digital Evidence, 1*(3).

Reno, J. (1996). Law Enforcement in Cyberspace Address. In D. E. Denning & P. J. Denning (Eds.), *Internet Besieged: Countering Cyberspace Scofflaws* (pp. 439-447): ACM Press.

Sato, O., Broucek, V., & Turner, P. (2005). Electronic Evidence Management for Computer Incident Investigations: A Prospect of CTOSE. *Security Management, 18*(3), 11-18.

Strauss, A., & Corbin, J. M. (1998). *Basics of qualitative research : techniques and procedures for developing grounded theory*. Thousand Oaks Sage Publications.

Urry, R., & Mitchison, N. (2003, May 8-9). *CTOSE Project. Electronic Evidence: gathering, securing, integrating, presenting.* Paper presented at the CTOSE Conference, Facultés Universitaires Notre-Dame De la Paix, Namur, Belgium.

Venema, W., & Farmer, D. (1995). SATAN (Security Administrator Tool for Analyzing Networks). Retrieved 10 January 2001, 2001, from http://www.porcupine.org/satan/

Verreck, P. (2000a). Case Study - Vindictive e-mail. Retrieved 25 November 2000, 2000, from http://www.forensic-computing.com/archives/vind.html

Verreck, P. (2000b). Presenting the Evidence. Retrieved 25 November 2000, 2000, from http://www.forensic-computing.com/archives/present.html

Other Papers and Contributions

A Testing Methodology for Anti-Spyware Product's Removal Effectiveness

Josh Harriman

Symantec Security Response, Dublin Ireland

About author

Josh Harriman is a Software Quality Assurance Engineer in Symantec Security Response. This group is based in Dublin, Ireland. Current duties include analysing viruses, and adware and spyware behaviour. He also performs testing of anti-virus and anti-spyware definitions and remediations produced by Security Response. Other responsibilities include research, competitor testing, and acceptance of future technologies of behalf of the Security Response Quality Assurance group. Josh has been working for Security Response since 2003. He previously worked in a quality assurance capacity for Stratus Computers and Into Networks in Boston, Massachusetts.

Contact details: Symantec Ltd. Ballycoolin Business Park, Blanchardstown, Dublin 15 Ireland, e-mail josh_harriman@symantec.com

Keywords

Anti-Spyware, methodology, testing, removal, adware, spyware, installation, detection, tools, effectiveness.

Abstract

Testing the removal effectiveness of an anti-spyware product relies on the tester having knowledge of the adware or spyware application, and having knowledge of the behavior it exhibits on the system under test. One must use various monitoring tools before, during, and after the test cycle, in order to really understand and measure how well the anti-spyware product removes adware or spyware from an infected system.

Some test results rely on interpreting the anti-spyware product's logs to determine how many risks were found and removed during a system scan. These logs are important, but they cannot be solely relied on to determine the removal effectiveness. Using other tools to monitor and log system behavior before and after a remediation by the anti-spyware product will help accomplish this task.

The tester doesn't need to be an expert in system analysis in order to conduct a thorough review, but they do need to take the necessary steps outlined in this paper to correctly report the removal effectiveness of an anti-spyware product. This paper will discuss the techniques needed to conduct an accurate and comprehensive evaluation of the removal effectiveness of any anti-spyware product in the market.

Background

Adware and spyware risks are as prevalent as ever. They are mentioned regularly in the media and even making it to the legal stage with a bill known as the I-SPY Act (U.S. House of Representatives, 2005). More and more users, both consumer and corporate, are finding these programs installed on their systems. In order to combat these risks, users need to have an anti-spyware product installed along with their anti-virus solution. Some AV vendors now combine both anti-virus and anti-spyware products into one. Other vendors are purely anti-spyware. While solid reputable tests for anti-virus products exist (Virus Bulletin, 2005), few are available for anti-spyware products. One positive testing method conducted for anti-spyware products can be found from Spyware Warrior (Howes, 2004). But since there are no standard testing methodologies for anti-spyware products, the consumer can receive mixed reviews. As Howes states, "Although a number of 'tests' of anti-Spyware scanners have been reported on the Net, many if not most of those tests are of limited value because the design, methodology, and execution of the tests is not fully and publicly documented, leaving even experienced users and experts to wonder just how meaningful those tests really are." (Howes, 2004, Overview p. 1)

Until a standardized testing methodology is created for anti-spyware products, reviewers will continue to conduct their own tests and produce results accordingly. One of the most popular reviews seen today is testing various anti-spyware products in a head to head battle to see which product comes out on top. These tests will be conducted on systems which have adware or spyware already installed on them. The anti-spyware products are then rated to see how well they removed the risks from the system. This is one of many areas that need to be looked at in detail in order to conduct a proper and thorough review of any anti-spyware product.

Current Methods

Most anti-spyware product testing falls into two methods.

1. Flat file scanning.
2. Follow-on scans by multiple products.

Flat file scanning is a testing method derived from anti-virus testing. This method is very easy to perform and provides quick results. A set of files deemed to be adware or spyware are placed on the test system and then scanned by the anti-spyware product. These files are not executed. But this type of testing also has its drawbacks. It does not test the true effectiveness of the anti-spyware product. Merely scanning a set of files will not test other important artifacts of adware and spyware. This is a non-holistic view of testing. You must look at the whole picture when dealing with adware or spyware. Registry keys, running processes, and other non-critical files will not be tested for removal with this method. Some products will have different detection techniques for adware and spyware. They might not detect a set of static files, but would actually detect and remove the risk if it were installed on the system.

Follow-on scanning is a testing method used to compare multiple anti-spyware products. A test system is first scanned with one anti-spyware product, then immediately following this scan, the test system is scanned by another anti-spyware product. This method also has various drawbacks. Many reviewers have used this type of testing to highlight how some products miss risks, which other products detect. This is misleading and not the correct method to evaluate the removal effectiveness. You cannot rely on any anti-spyware product to inform you of which files are clean, or which registry keys should or shouldn't be removed. Just because one vendor's product reports a risk and removes some set of files and registry keys, doesn't mean that they were correct in doing so. You

need to know which files, processes, and registry keys are part of the risk, and which ones are possibly clean. Don't rely on the product to tell you.

Neither of these methods is effective. Other anti-spyware product testing can consist of installation, performance, usability, and prevention. This paper will not discuss these areas of evaluation. In order to know which product removed the risks properly and effectively, the tester needs to take a more detailed approach in understanding the adware and spyware applications.

Testing Methodology

In order to evaluate how well an anti-spyware product has removed an adware or spyware risk, you must be able to monitor exactly what was installed by the risk onto the test system. Some adware or spyware risks are fairly simple applications and it's easy to monitor their installations and behavior. Others can be quite complex and very difficult. You can read more about the details of adware and spyware techniques from the Virus Bulletin Conference proceedings (Chien, 2005).

It is also important to understand that some files and registry entries will be made by the system itself as a result of installing or executing some risks. Some of these files and registry entries need to be treated differently. Also, some files and registry entries will be shared components. These shared components might belong to legitimate programs and should not be removed. The changes made by the system and legitimate programs will be discussed further in the areas of interest section.

Tools

You will need to use various monitoring tools to capture changes made to the test system by the adware or spyware risk. Most of the tools mentioned are freely available on the web.

- **Regshot** - This is a freeware utility that monitors changes to the file system and registry. It will monitor files and registry entries that were added, deleted, or modified. It's small and fast, but will not save file system changes if you need to reboot in between system snapshots.

- **WhatChangedForWindows** - A complete tool for file and registry monitoring. Has the ability to track changes during reboots. Can also be run as a Management Console to monitor multiple clients. Can be found here: http://www.prismmicrosys.com/whatchanged/index.htm
- **Filemon** - A file system monitor from Sysinternals which can be found here: www.sysinternals.com. Monitors activity in real-time.
- **Regmon** - A registry monitor from Sysinternals which can be found here: www.sysinternals.com. Monitors activity in real-time.
- **ProcessExplorer** - A process monitor from Sysinternals which can be found here: www.sysinternals.com. Monitors running processes.
- **RootKitRevealer** - A root kit detection utility from Sysinternals which can be found here: www.sysinternals.com. Can be used to detect various forms of spyware that try to hide from the system and user.
- **Ethereal –** A network protocol analyzer. An open source tool available from www.ethereal.com. Can be used to determine exactly where various forms of adware and spyware are coming from on the Internet.

Test environment

To ensure a repeatable testing environment, the test system should be imaged. Some imaging products available are Norton Ghost which can be found here: http://www.symantec.com/home_homeoffice/products/backup_recovery/ghost10/index.html or VMware which can be found here: www.vmware.com. Be aware that some risks have the ability to determine if they are running in a VMware session. This could result in the risk not running at all, or give false results as to its behaviour. Using an imaged system is important for repeatable testing across multiple anti-spyware products. The Operating System choice and setup of the test system should be similar to that of a typical user. A good base setup would be Windows XP Home Edition with Service Pack 2. This would suffice for most consumer users. If the testing is also to be conducted for the corporate edition of an anti-spyware product, then other types of Operating Systems might need to be considered. Install the monitoring tools listed above and create a clean test image.

Step-by-step testing

The following steps should be used when conducting the removal effectiveness of an anti-spyware product. As stated previously, this test method is only to evaluate the

removal of the adware or spyware risk which was installed on the test system. Before starting these steps, decide which risks you want to install. The adware and spyware landscape is constantly changing. Some risks update themselves on a weekly basis. Others might stay static for months. Try to gather the most widespread risks for testing. This will ensure good coverage for the typical user. Choosing one risk or many risks is irrelevant, but what is important is that you record the location, name, MD5, and a copy of the actual file used during testing. This will be important in regards to the validity of the testing. The information gathered from the steps below will be used in the results phase of the testing methodology.

1. Begin with a clean imaged test system. This machine needs to be connected to the Internet.

2. Using one of the system change monitoring tools, take a baseline snapshot. This snapshot should include the file system and the registry. Save this snapshot for comparison during testing.

3. Start the real-time system monitoring tools. These include Filemon, Regmon, ProcessExplorer and Ethereal. The logs from these tools can help match various files and registry entries to the actual risks installed.

4. Install the adware or spyware applications. Various web sites have information about these risks and some links to locations where they can be installed (spywareguide, 2005). If you use Windows XP with Service Pack 2, you will need to accept the ActiveX controls used for installation by certain risks. Some risks will install or bundle other risks. You will need to wait some small amount of time, possibly 15 minutes, to give the risks time to properly install.

5. Stop real-time monitoring tools and save the logs.

6. Reboot the test machine. Some risks need to have a system reboot for full installation.

7. After reboot, wait again for a few minutes before taking the second snapshot with the system change monitoring tool. This snapshot will list the adware or spyware file and registry modifications. This information will be used to show what was added or changed by the adware or spyware risk.

8. Create and save an image of the test system. This imaged system will be used for testing against multiple anti-spyware products. Before testing against one anti-spyware product, restore the system to this image, and then install the anti-

spyware product for testing.

9. Install one anti-spyware product for testing. Make sure to update the product and anti-spyware definitions to the most recent versions. Make note of the product and definition versions. These will be used during the results phase.

10. Start *Internet Explorer* (IE), as some adware or spyware applications are loaded into this process. Run a full system scan with the anti-spyware product. If any alerts pop-up from the product e.g. denying access from suspect programs, then always choose YES to deny. This will help limit the installation of further risks.

11. If the anti-spyware product requires a reboot to complete the remediation, then reboot before taking the final system snapshot.

12. Take the final system snapshot. This snapshot will be compared with the one taken during Step 7. The comparison will show what files and registry entries were either removed, added, or modified during remediation.

13. Take note of any problems or issues with the test system after the full remediation, such as system crashes, blue screens, etc… Start *Internet Explorer* (IE) and make sure the system still has Internet connectivity. If the adware or spyware risk was installed as a layered service provider, an improper removal will cause loss of network connectivity. LSP installs will be discussed in the areas of interest section. Also watch for any error dialog boxes or messages, and any ads that are still being generated. Some anti-spyware products will report they have successfully removed a risk and its associated files and registry entries. But in fact the risk either re-installed itself on the system or was never removed at all.

Areas of interest

Files and registry entries will be discussed as areas of interest as they are needed to formulate the results. These will be gathered during Steps 4 – 7 above.

Load points

Windows load points are used by adware and spyware applications to ensure they run each time the computer starts. The anti-spyware product must handle these locations

during remediation. These load point areas should be highlighted during the results phase, as they are important factors to determine the removal effectiveness. Another point to note with load points is that some anti-spyware products might successfully remove the file, but miss the load point. This will most likely cause simple *Windows* error messages to pop up after every reboot. Below is a list of some common load points used by adware and spyware risks.

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
    - The most common load point in *Windows* is the Run key in the registry. Files located in this key will execute after system startup.

Another common area targeted by adware or spyware applications is to load their programs into *Internet Explorer* (IE) as toolbars or browser helper objects.

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects
    - "In its simplest form, a BHO is a COM in-process server registered under a certain registry key. Upon startup, Internet Explorer looks up that key and loads all the objects whose CLSID is stored there." (Esposito, 1999)

- HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Toolbar

- HKEY_LOCAL_MACHINE\Software\Classes\CLSID\<CLSID number>\InprocServer32
    - A path to the file, which is loaded as a COM object, will be located in this key.

A sample log generated with the Regshot tool while monitoring the installation of a browser helper object and toolbar from Step 4 using a sample from IEPlugin is show in (Figure 1) below.

```
Keys added:468
...
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Browser Helper Objects\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}
...

Values added:596
...
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\VersionIndependentProgID\: "IMIToolbar.imiTool"
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\TypeLib\: "{58D419E8-1321-4DD2-A6FC-7B41C14DCD79}"
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\ProgID\: "IMIToolbar.imiTool.1"
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\InprocServer32\: "C:\WINDOWS\systb.dll"
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\InprocServer32\ThreadingModel: "Apartment"
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}\: "Intelligent Explorer"
...
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Internet Explorer\Toolbar\{69135BDE-5FDC-4B61-98AA-82AD2091BCCC}:
...

Files added:457
...
C:\WINDOWS\systb.dll
...
```

Figure 1: Log file from Regshot tool showing the installation of a Browser Helper Object and toolbar.

Some other load point locations are listed below.

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Winlogon
  - The *Shell* value by default will have Explorer.exe as the process that loads during system startup. Some risks will add a path to their file in this location.
  - The *Userinit* value by default will have userinit.exe as the process that loads during system startup. Some risks will add a path to their file in this location.

- HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
  - Another location in the registry that will load files after system startup.

- HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Runonce

- HKEY_CLASSES_ROOT\exefile\shell\open\command
  - The *(Default)* value could be changed to load a suspect file every time an .exe file is executed on the system.

Files

Files usually fall into two broad categories of critical or non-critical. And within these two categories you will have both clean files and detectable files. Critical files are considered to be executable e.g. EXE, COM, or CAB files, and dynamic link library e.g. DLL files. The executable files will usually be running as process or service. Their functions might be to download other files, load DLL files into various processes, or contact a particular Web site to check for updates. Non-critical files can consist of logs, configuration, and information e.g. readme text files.

You can monitor files with the system change tool and Filemon. These logs will show what files have been added and by which process. ProcessExplorer will show not only the running processes, but also the modules running inside another process. Using the IEPlugin sample again we can see that within the *Internet Explorer* (IE) process, the file systb.dll is loaded as a module and running under the context of (IE) (Figure 2).

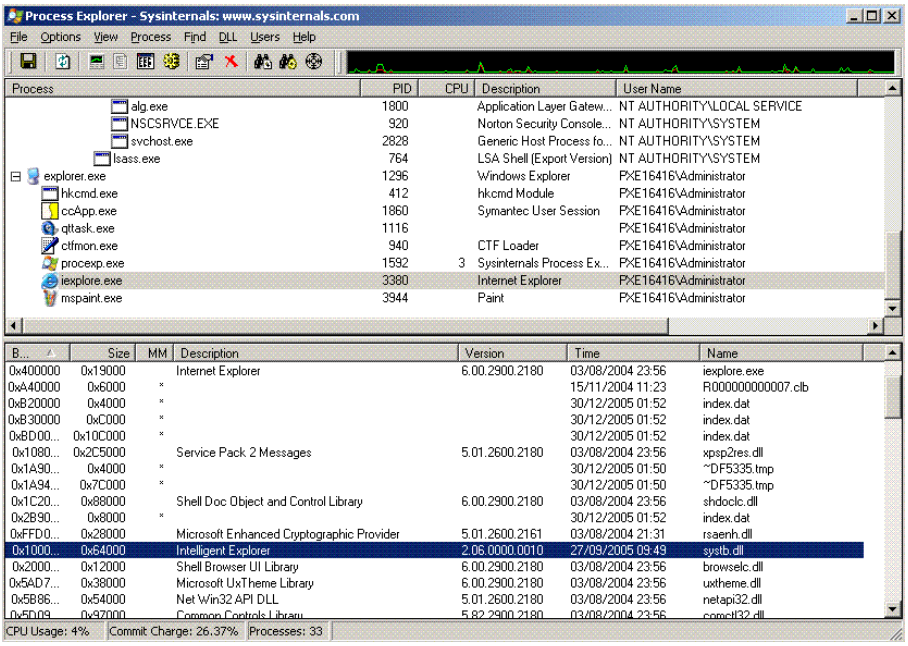


Figure 2: IEPlugin DLL file loaded and running in IE process.

When surfing the Web, various files will be downloaded. They will be located in the Cache directory for the browser. For *Internet Explorer* (IE), the path to this folder is usually located in C:\Documents and Settings\<username>\Local Settings\Temporary Internet Files\Content.IE5. Under this folder are sub-folders that are randomly named e.g. 4TU30D6R. All the files from Web surfing with (IE) will be in these sub-folders. This could include .CAB, .EXE and .DLL files. Note that some anti-spyware products might have a separate function to completely clear the Cache directory. Others might delete various non-critical files from these locations such as .GIF, .JPG, .HTM, and others even clear the reference of the files from the index.dat file.

Clean files are those that belong to a legitimate application or the Operating System. Some adware and spyware risks come bundled with legitimate programs. They could share a common component, which both applications might need in order to function properly. Most of these files will be located in the parent directory of the application that was installed but could exist in the %WINDOWS% or %SYSTEM% directories. For example, one clean file which should not be removed is %SYSTEM%\unwise.exe. This file is used to uninstall applications that were created with the *WISE* installer. Deleting this file could cause legitimate applications problems when trying to uninstall them. Some adware or spyware risks might include an unwise.exe file, but place it in their own directory.

Prefetch files are files that *Windows XP* uses to speed application startup procedures. When adware or spyware applications are installed on a *Windows XP* system, a .PF file

will be created in the %WINDOWS%\Prefetch directory. More information about prefetch files can be found in the Microsoft Developers Network magazine. (Russinovich & Solomon, 2001) You will notice that some anti-spyware products will detect and remove prefetch files. This is not really necessary, as the system will remove these files when they are no longer needed. They also do not pose a security risk on your system as they cannot be executed. But if an anti-spyware product does delete them that is ok as well.

Registry

Many registry entries will be added or modified by the installation of an adware or spyware application. Even the most basic application might modify hundreds of registry keys and values. If the adware or spyware application is bundled with a legitimate program e.g. peer-to-peer or bundled with other adware or spyware applications, then the number modifications could be well over one thousand.

Some registry keys are created by the system as an indirect result from installing some adware or spyware risks. One of the most common registry keys to be added on a *Windows* XP system is a CLSID key located under HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Ext\Stats\.

This key is not well documented, but is holds information of ActiveX installs used by *Internet Explorer* (IE).

Some adware or spyware applications will load into the *Windows* layered service provider chain of Winsock. Winsock is needed for all network connectivity and if an entry from the LSP chain is removed improperly, then network loss is possible.

As mentioned in the load point section above, BHO's and toolbars create a key or value as CLSID type number. This key is used to find the COM object's information located in HKEY_LOCAL_MACHINE\Software\Classes\ CLSID. While it's important for the BHO and toolbar entries to be removed, the actually CLSID key itself cannot load the COM object and therefore is not as critical. It should be removed though if it's certain the key is part of the adware or spyware risk.

Some adware or spyware risks will alter various registry keys, which control your browser home or search pages. It would be impossible to know what the original settings of these keys were before infection, so there is no real proper reset value. But they should be cleared of any reference of the risk.

Special cases

Some adware or spyware risks will try and hide themselves from the user and from the system. Some versions of Elitebar do just that and are known as user-mode rootkits. Some anti-spyware products will handle these rootkits, while others will fail to even detect much less remove them. You can view some rootkits with the RootKitRevealer tool. If you encounter some of these risks during testing, make sure to include a note about a risk that is hidden. Also, be aware that some anti-spyware products will remove these risks, and even the system analysis tools will report the removal, but the files will be recreated after the full remediation. This information needs to be recorded during the reporting phase.

Results

Reporting the correct results gathered from the step-by-step testing above a very crucial element of the testing methodology. It's important to note that not all anti-spyware products will remove the same set of risks. This could be a limitation of technology, or could also be a company policy with regards to a particular risk. This could prove difficult when reviewing multiple anti-spyware products.

Another ineffective testing method is to use one anti-spyware product's result to gauge another product's effectiveness. This method should never be used. Even using the logs reported by one anti-spyware product is the wrong way to conduct the results phase. Just because a product states that it has successfully handled or removed the adware or spyware risk, you must use the system monitoring tools to verify this.

Some mistakes made in product reviews of this nature include the assumption that the anti-spyware product will properly remove an adware or spyware risk based upon the detection only. Full remediation must be run on the anti-spyware product to verify the removal. Some anti-spyware products show all the associated files and registry entries for a particular risk during the detection and removal actions whereas other products might only show one file during detection, but actually remove the same set of files and registry entries during the removal action. So it is important to use the system monitoring tools to see exactly which files were removed and which registry entries were removed or modified.

Reporting the findings

All of the detail that goes into following the step-by-step testing and the areas of interest are not needed in the final report. The report needs to be concise yet complete. A good report should include the following:

1. Common name of adware or spyware samples used for testing.
    a. Most adware and spyware risks have common names that are shared among the anti-spyware industry.

2. Files and registry entries added or modified by the installation of the adware or spyware sample.
    a. These files can be grouped together as totals into meaningful buckets for critical and non-critical. The same could be applied to the registry keys and values following the load point section listed above.

3. Product information.
    a. The exact version of the anti-spyware product.
    b. The exact version of the data file definitions. If versioning information is not available, then the exact date and time the definitions were updated.

4. Files and registry entries deleted or modified by the anti-spyware product.
    a. These files can be grouped as totals into meaningful buckets for critical and non-critical. The same could be applied to the registry keys and values following the load point section listed above.
    b. This could also be represented as a percentage.

5. Any issues with the test system or applications after the full scan is completed.
    a. This would be detail from Step 13 in the step-by-step testing section.

6. An appendix that list details of the adware or spyware samples.
    a. This would include the location of the sample e.g. the Internet address, the MD5 value of the samples, the date and time each sample was harvested, and if the sample was gathered from a bundled legitimate program such as a peer-to-peer program.

A rating scale could be used to show the removal effectiveness of the anti-spyware products.

1. Poor – Not enough critical files, processes, load points removed. Adware or spyware risks still exist on the system.

2. Good – All critical files, processes, load points removed. All adware or spyware risks were neutralized on the system. Some remnants are left behind. These remnants could be ext\stats keys, log files, empty directories.

3. Best – All critical files, processes, load points and remnant side effects are removed. This would be a complete removal of all files and registry entries created by the adware or spyware risk. This would also include some system files and registry entries such as the ext\stats keys and prefetch files.

4. Aggressive – This rating could be given to an anti-spyware product that removed too much from a system or incorrectly removed some risks. This would include deleting %SYSTEM%\unwise.exe, incorrectly removing LSP registry entries, and deleting shared component files or registry entries of a legitimate application if their removal caused system instability, or application crashes.

Buckets

Reporting that an anti-spyware product removed 80% of risks from the system, as a stand-alone comment, is not very meaningful. 80% might sound good, but if the other 20% was some load point in the registry or critical .DLL file in the %SYSTEM% directory, then the result is not that good. After gathering the information from Steps 4 – 7 in the step-by-step section above, the file and registry entries should be separated into meaningful buckets.

Files could be listed as follows:

- Critical – All .EXE, .DLL, .CAB, .COM
- Non-critical – All .TXT, .GIF, .HTM, .HTML, .DAT, .LNK, .PF, image files (.JPG, .GIF, .BMP), directories created by the risk.

Registry entries could be listed as follows:

- Critical – All load points such as Run key, BHO's, toolbars, Explorer shell value, browser home and search pages.

- Non-critical – All ext\stats keys, CLSID key in the root Classes hive, risk named key under the Software hive e.g. HKEY_LOCAL_MACHINE\Software\AdwareName

Another bucket could be for processes and services. Even though the process or service will be run by an EXE, it is a critical part of the removal action of an anti-spyware product and should be noted separately.

Conclusion

Testing the removal effectiveness of an anti-spyware product properly is not an easy task. Various steps are required to ensure the tests are conducted thoroughly. First and foremost, the anti-spyware product's logs should never be used to tabulate the results. Also, running one anti-spyware product immediately after another anti-spyware product to see what the first one missed is not a good indication of the effectiveness of either product.

Gather all the information from the test system before, during, and after the installation of the adware or spyware risk. Use that information as a base to judge all of the anti-spyware products individually. Report the findings clearly and if any issues arise, such as why one product didn't remove some adware or spyware, contact that company and see if there is policy for that particular risk. Don't just assume it was missed.

Make sure to run a full system remediation from the anti-spyware product. Do not report the results on detection alone. Removal is the key to this testing methodology and should not be limited to detection numbers and how many risk were found on the system. Since some anti-spyware products will remove the same set of risks, but have different methods of detection, the detection numbers can be very misleading. This testing methodology should be used to determine the removal effectiveness and that can only be accomplished by following the steps outlined in this document.

References

U.S. House of Representatives (2005). Internet Spyware (I-SPY) Prevention Act Of 2005. Retrieved January 15, 2006 from http://thomas.loc.gov/cgi-bin/cpquery/R?cp109:FLD010:@1(hr093)

Virus Bulletin (2005). Independent comparative testing of anti-virus products known as the VB 100% award. Retrieved January 17, 2006 from http://www.virusbtn.com/vb100/about/index.xml

Howes, E. (2004). The Spyware Warrior Guide to Anti-Spyware Testing. Retrieved January 17, 2006 from http://spywarewarrior.com/asw-test-guide.htm

Chien, E. (2005). Techniques of Adware and Spyware. Proceedings of the 15th Virus Bulletin International Conference. Dublin, Ireland.

Spywareguide (2005). List of Adware and Spyware products found on the Internet. Retrieved January 15, 2006 from http://www.spywareguide.com/product_list_full.php.

Esposito D. (1999). Browser Helper Objects: The Browser the Way You Want It. Retrieved January 17, 2006 from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebgen/html/bho.asp.

Russinovich, M., & Solomon, D. (2001). Windows XP: Kernel Improvements Create a More Robust, Powerful, and Scalable OS. Retrieved January 20, 2006 from http://msdn.microsoft.com/msdnmag/issues/01/12/XPKernel/default.aspx

Appendix A

MD5 value of sample used for *Figure 1 & Figure 2*

IEPlugin – systb.dll 2bb2031ba6ae4d595976b9ca4201ffc6

Behavioural Classification

Tony Lee & Jigar J. Mody
Microsoft Corp.

About Author(s)

Tony Lee is a lead AV researcher in Anti-malware team at Microsoft. His current research focuses on automated malware analysis and heuristic detections. He graduated from University of California at Berkeley with BS in Computer Science and Electrical Engineering, and later MS from University of California at Los Angeles, where he did his research on query relaxation, knowledge database, and XML-based data clustering.

Contact Details: Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399, USA, phone 1-425-722-0590, fax 1-425-936-7329, e-mail tony.lee@microsoft.com

Jigar J Mody is a Program Manager in Anti-malware Team at Microsoft Corporation. His current research focus is on developing Machine Learning solutions to aid the process of automated malware analysis and heuristic detections. He holds a Master in CS from Arizona State University (ASU) where his focus of research was committee based learning. He was a teaching assistant at ASU for graduate Data Mining course.

Contact Details: Microsoft Corporation, One Microsoft Way, Redmond, WA 98052-6399, USA, phone 1-425-705-7205, fax 1-425-936-7329, e-mail jigarm@microsoft.com

Keywords

Malware, classification, data mining, machine learning, clustering, similarity, behaviour, event

Behavioural Classification

Abstract

In recent years, the number of malware families/variants has exploded dramatically. Automatic malware classification is becoming an important research area. Virus/spyware writers continue to create large number of new families and variants at an increasingly fast rate, effectively rendering manual human analysis inefficient and inadequate. In attempts to automate static file analysis, we encountered considerable challenges from incremental family evolutions, binary obfuscation and intricate component relationships associated with Spyware. These challenges suggest the importance of run time behaviour analysis in addition to static binary analysis, and using adaptable algorithms to automate classification. In this paper, we propose a behaviour-based automated classification method based on distance measure and machine learning.

Introduction

The dramatic growth in malware prevalence in recent years has challenged the Anti-malware industry and researchers in areas ranging from detection coverage, to effective threat removal, classification, analysis/description, etc. In this paper, we will mainly focus on classification problem; on the other hand, we see that an effective classification method can serve better detection, cleaning and analysis solutions.

The process of classifying malware samples into families is mainly driven by the following steps,

1. Analyse an Object (sample).
2. Represent and store knowledge of the object in a structured.
3. Reference learned knowledge, apply classifiers to recognize familiar patterns and correlate similarities.

Figure 1 below describes this classification process,

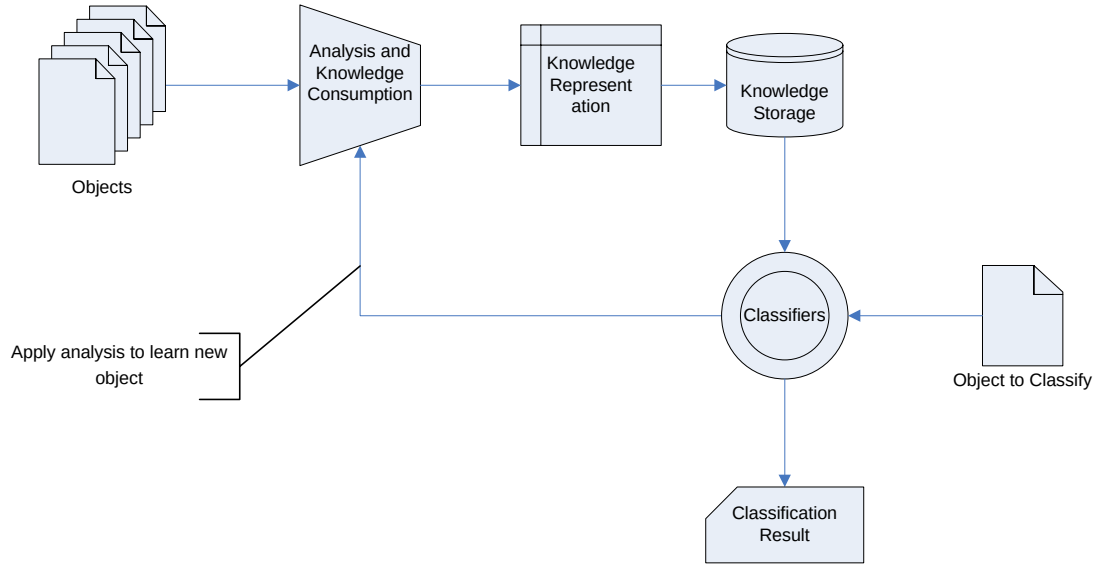


Figure 1: Classification Process

In practice, we are also concerned with the time and storage complexities of the classification process. Therefore, we can reduce the classification problem to how we can effectively and efficiently capture, structure, store, and apply knowledge.

Traditionally, when new malware samples are discovered, human analysts classify these samples by memorization, looking up description library or searching sample collection. Human analysis is time consuming, subjective, and results in considerable information loss. The industry and research community have made attempts, both in implementation and study, as further discussed later, to address some of these challenges aforementioned, but in a partial and isolated manner. We have taken a holistic approach to tackle the classification problem, from knowledge consumption, representation and storage, to classifier model generation and selection. We set forth to develop classification methodologies targeting the following objectives,

- The process consumes knowledge about the object (sample) efficiently and automatically.
- The knowledge is represented in a form that results in minimal information loss.
- The knowledge is structured to be stored, analysed and referenced efficiently.
- The process effectively and automatically applies learned knowledge to identify familiar pattern and similarity relations in a given target.
- The process is adaptable and has innate learning abilities.

In this paper, we propose an automated classification method based on runtime behavioural data and machine learning. Specifically, we represent a file by its runtime behaviour in the form of sequenced events. We structure the event information in a canonical format and store them in database. We construct classifiers for machine learning, which take the event data as input and recognize patterns and similarities across large quantities of objects. Finally, the similarities and patterns learned by the classifiers are applied to classify new objects.

Related Work

The IBM Digital Immune System was among the first to examine data mining techniques applied on static binary analysis [1]. DIS also suggested the possibility of run time analysis via emulation.

Historical statistical analysis on API calls was used to detect behavioral anomalies [2,3,4]. A monitor agent logs process activities and discerns degree of malice by accumulative scores in the form of weights assigned to API calls. This API weighting scheme captures only a simplistic view of object behaviour; the weights and threshold are difficult to determine and may not converge for all types of behaviours. Also the weighting scheme is designed to block malicious processes in real time, and not intended for classification beyond that between malicious and clean.

Additional research on classification [5] built statistical models of behavior to calculate pair-wise similarities between the behaviors of two objects, using probabilistic suffix tree and Makovian statistics. However, these algorithms only work well on inferring deviations, but relatively ineffective in comparing two unrelated objects. These algorithms thus are not well suited for general classification purpose. No attempts were made to apply data mining using these similarity measures.

Later research [6] explored the extraction of malware behavior features and the application of data mining to classification. While the algorithms employed were adaptable, the abstract feature set approach does not capture the ordering of actions, nor can it scale automatically for new and/or unknown behavior features. Consequently, the method only works well to detect general known

category of malwares (e.g. Trojan droppers, Trojan Spy), but cannot scale to general purpose classification on a malware family level.

Knowledge Representation

The problem of knowledge representation plays important role in the accuracy and efficiency of the classification process. A good knowledge representation has the following properties:

- Effectively capture knowledge of the object to represent; the representation results in minimal information loss.
- The representation can persist in permanent storage. The representation can be efficiently referenced and searched.
- The representation can enable classifiers to efficiently and effectively correlate data across large number of objects.

We construct knowledge representation of malware files in order to classify these files into families based on their knowledge representation. Malicious software is classified into families, each family originating from a single source base and exhibiting a set of consistent behaviors. Characteristics of a malware are encoded in its binary structure and functional semantics expressed in its code and data.

Static Analysis vs. Runtime Analysis

Static analysis primarily targets the structural information of a file. Structural comparison (e.g. by call graphs and code blocks [8,9,10]), measures file similarities in terms of quantity of variation, but largely overlooks the semantics (quality) of the variation. For example, the Win32/Mytob worm evolved from Win32/Mydoom into a separate family due to an additional exploit infection vector, but the resulted structural variation is fairly small. IRC bot family evolutions [7] also present similar phenomena due to code sharing. Static analysis can heuristically decipher some functional semantics by analyzing API calls, data, and code structures, but falls short of fully understanding functional semantics, which is a more natural problem to solve by runtime behavior analysis.

Static analysis often fails to reveal inter-component/system interaction information. Typical example is spyware which often comes in packages and leverages multiple components working collaboratively. Runtime analysis has distinct advantage to capture such dynamic knowledge.

Code/data obfuscation (e.g. encryption, packing and polymorphism/metamorphism) also poses considerable challenges to static analysis in practice, for example, in the case of on-access obfuscation (i.e. a process decrypts instructions when they are executed and re-encrypts them after execution; data is only un-obfuscated when being used). In many case, runtime analysis can effectively bypass the need to deal with these obfuscations.

These challenges aforementioned suggest the importance of using data from runtime analysis in addition to static analysis to achieve optimal classification results.

Representing Behavior

Representing behavior information is a non-trivial problem. Monitoring API call history was used to discover program behavior [2,3,4]. High level functional attribute sets were applied to classification [6]. These methods invariably result in considerable information loss, because a complete representation must be aware of the following inherent properties of program behavior:

- Ordering with respect to time
- Actions taken by the subject (program)
- Subsequent state of the environment in which the action takes place.

The meaning of a particular action taken by a program at a particular time must be conceived in the context of resulted state of the environment and the actions/environment states previously take place. Furthermore, for the purpose of classification automation, we want to construct the representation in a consistent canonical format, such that classifiers can consume the information to effectively recognize patterns and correlate similarities across large number of objects.

The Vector Approach

Classification can efficiently process data in vector format using statically and probabilistic algorithms, which are generally preferred approaches. The challenges of feature extraction we have encountered as we have also seen in other problem space are due to the complexity of the behavior knowledge. We are always working to strike the difficult balance among vector size, scalability, and factorability.

- Size: if we represent information solely by binary digits, 1s and 0s, it is most general with near zero information loss, but the size of the vector representation is infinite.
- Scalability: if we represent information in a list of predefined high level concepts (e.g. password stealing, backdoor, email, etc.), we can construct a neat vector of acceptable size; however, this approach cannot scale to unaccounted or unknown behavior types.
- Factorability: general concepts such as password stealing is very informative, but also very difficult, if not impossible, to factorize (enumerate) all possible facts that support the concept. The result is likely that we will have false positives and negatives when abstracting the concept from collected facts.

The complexity in extracting features out of a malware is a difficult task. We decided to take another approach.

The Opaque Object Approach

Another approach to represent classification data, as we have taken, is use opaque object. In this approach, we do not attempt to tabularize data, instead allowing the objects to represent data in rich syntax (e.g. hierarchical, sequenced, etc.) We define distance measure that yields similarity distance between any two given objects. In this way, we lost the ability to construct statistical / probabilistic views of the data, but it is an acceptable trade off given the complexity of the knowledge; however, we achieve rich semantic representation of the actual object (behavior) and precise distance between objects, which will serve basis to classification methods based on clustering, later described.

Events Representation

The opaque object we use to describe behavior consists of a sequence of events, because events are the basic elements that define actions and state transitions in an event-driven system such as the Windows Operating System. The event sequence is ordered according to the time of the occurrence of program actions and environment state transitions.

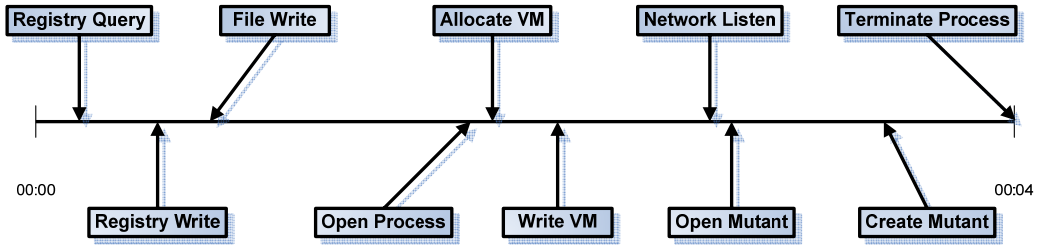


Figure 2: An Example of a Event Sequence

To capture rich behavior semantics, each event contains the following information

- Event ID
- Event object (e.g registry, file, process, socket, etc.)
- Event subject if applicable (i.e. the process that takes the action)
- Kernel function called if applicable
- Action parameters (e.g. registry value, file path, IP address)
- Status of the action (e.g. file handle created, registry removed, etc.)

For example, a registry open key event will look like this,

Tag	**Value**
Event ID	8
Event Object	Registry
Kernel Function	ZwOpenKey
Parameter 1	0x80000000 (DesiredAccess)
Parameter 2	\Registry\Machine\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\iexplore.exe
Event Subject	1456 (process id), image path \Device\HarddiskVolume1\Program Files\Internet Explorer\IEXPLORE.EXE
Status	Key handle

Table 1: Registry Event

FROM CLASSIFIER TO CLASSIFICATION

Machine learning is a method to investigate the mechanisms by which knowledge is acquired through experience. Classification is one of the most used machine learning methods. Classification is a form of supervised learning, which requires training data, with known input/output, to form priori knowledge. The task of the supervised learner is predict the value of the classifier function for any valid input after having seen a number of training examples. The process of classification consists of the following tasks:

1. Data pre-processing
2. Model generation
3. Model selection
4. Classifier generation

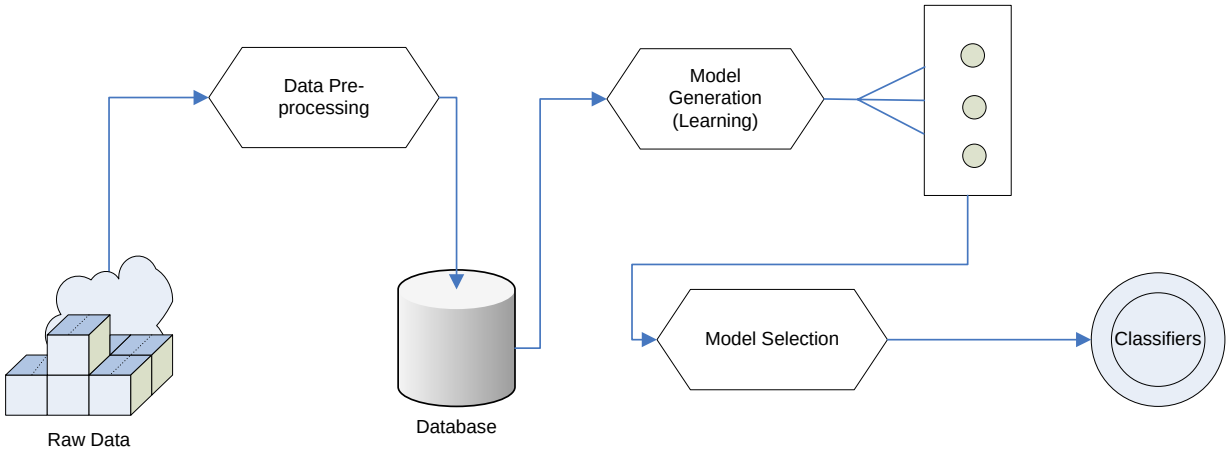


Figure 3: Generate Classifier for Classification

Common classifiers include probability-based, statistics-based and case-based classifiers. Because event data is ordered and non-tabular, we use case-based classifier by treating existing malware collection as a database of solutions. Case-based classifiers learn by CBR (case based reasoning), a process of solving new problems based on the solutions of similar past problem. CBR is a prominent kind of analogy making. Commonly used in CBR solutions is Nearest Neighbor algorithms. In Nearest Neighbor, we compare an object to all known objects (training data) and find the known object that is most similar (nearest) to the new object; if this minimum distance is within acceptable range, the new object is then assigned the same class label as that of its nearest neighbor.

Clustering Overview

While the CBR approach may sound intuitive, it is non-trivial to make it scalable. To address scalability issue, we apply clustering, a machine learning technique, to form a specified number of clusters, for which a representative object is selected to represent its clusters.

Clustering is a form of unsupervised learning, a process to organize objects into groups whose members are similar in some way. Clustering discovers hidden semantic structure in a collection of unlabeled data. A cluster is a collection of objects which are “similar” between them and are “dissimilar” to the objects belonging to other clusters. Mathematically, clustering is the partitioning of a data set into subsets (clusters), based on a defined distance measure, such that the data in each

subset is closest to each other, and furthest from those in other clusters. Ideally, data in each cluster should share some common traits.

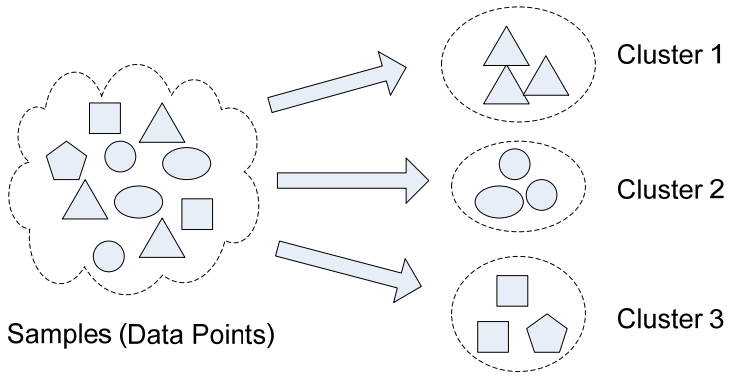


Figure 4: Clustering of Shapes

Distance Measure

Essential to clustering is the distance calculation that measures similarity between objects. Clustering distance measure D(obj1, obj2) has the following properties:

1. Distance is non-negative: D(obj1, ojb2) >= 0
2. A point is distance 0 from itself: *D(x, x) = 0*
3. Distance is symmetric: *D(x, y) = D(y, x)*
4. The triangle inequality: *D(x, y) < D(x, z) + D(z, y)*

Tabular data points are often thought to live in a K-dimensional Euclidean space. However, for non-tabular data, these distance measures do not apply, therefore we cannot pinpoint data points in K-dimensional space. Some examples of clustering without Euclidean space include clustering of color objects, text documents and ordered sequences, in our case.

Determining distance measure between ordered sequences is a non-trivial problem. Similarity is encoded in the content of each node in the sequence and the ordering of the nodes. We adapt Levenshtein Distance, commonly known as string edit distance, to measure event sequence similarity, because there exists a close analogy in the two problem spaces. The generalized Levenshtein Distance is defined by the minimum cost required to transform one sequence of objects to another sequence by applying a set of operations.

Specifically, we define an operation by applying one of the operators on the event(s) in the sequences,

Operation = Op (Event)

Operations include but not limited to

1. *Insert (Event)*
2. *Remove (Event)*
3. *Replace (Event1, Event2)*

The cost of an operation is defined by the cost of applying an operator on event(s)

$$Cost\ (Operation) = Cost\ (\ Op\ (Event))$$

The cost of a transformation from one event sequence to another is defined by the cost of applying an ordered set of operations required to complete the transformation.

$$Cost\ (Transformation) = \sum_i Cost\ (Operation_i)$$

The similarity between two event sequence is therefore minimum cost incurred by one of the transformations.

To further refine the similarity measure, we adjust the cost of each operation based on the operand (event) of the operation. Cost adjustment allows us to refine distance measure from training data, therefore increasing the accuracy of the clusters. As a simplified example, we can define the following cost matrix shown in Table 3

Operator	Event Object 1	Event Object 2	Cost
Insert	Registry – Write		1.5
Insert	File – Write		1.8
Remove	Process – Create		0.7
Replace	Registry – Write	Registry – Write	0.1
Replace	Registry – Write	Registry – Read	0.4
Replace	File – Write	File – Read	0.4
Replace	Network – Write	Network – Read	2.0
*	*	*	1

Table 3: Operation Cost Matrix for Similarity Measure

K-medoid Clustering

Hierarchical and partitioning clustering are two common approaches to general purpose clustering. Hierarchical clustering algorithms group data objects into a hierarchy (e.g., a tree) of clusters. These algorithms generally do not scale well with the number of data objects. The computational complexity is usually O(n2) [14,15]. Partitioning algorithms partition objects into k clusters [15], where k is the number of desired clusters, $k<=n$. The clusters are formed in order to optimize an objective criterion such as distance measure. Each object is assigned to the closest cluster. Clusters are typically represented by either the mean of the objects assigned to the cluster (k-means [12]) or by one representative object of the cluster (k-medoid [11]).

Because we are primarily concerned with family level classification, and our purpose is model generation (i.e. cluster representative selection), we choose to apply partitioning clustering based on

adapted Levenshtein Distance measure described previously. We subsequently select the k-medoid partitioning clustering algorithm, a modified version of k-means, because clustering in our case is based on data points and not mathematical values (i.e. based on medians in place of means). The goal of the algorithm is to divide the objects into k clusters such that the similarity measures to the cluster centroids (most centered data point) are minimized. Specifically, we determine k the number of families in experiments in order to evaluate the effectiveness of the event based representation.

Step	Description
1	Place K points into the space represented by the objects that are being clustered. These points represent initial group medoids.
2	Assign each object to the group that has the closest medoid
3	When all objects have been assigned, recalculate the positions of the K medoid.
4	Repeat Steps 2 and 3 until the medoid no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Table 4: K-medoid algorithm

The medoids of each cluster are the best representatives of their respective clusters. We select these medoids as our representatives (classification model).

Classification

Given classification model (i.e. cluster representatives), we perform the nearest neighbor classification, an effective method for classifying data objects similar to training data objects [13]. A new object is compared to the medoids based on the adapted Levenshtein Distance measure. The class determination for the sample is set as the same as the class of the nearest medoid.

Step	Description
1	Compare the new object to all the medoids
2	Assign the new object the family name of the closest medoid

Table 5: Nearest Neighbor Classification

EXPERIMENTAL RESULTS

Data Collection

In our experiments, we use an automated distributed replication system developed internally at Microsoft.

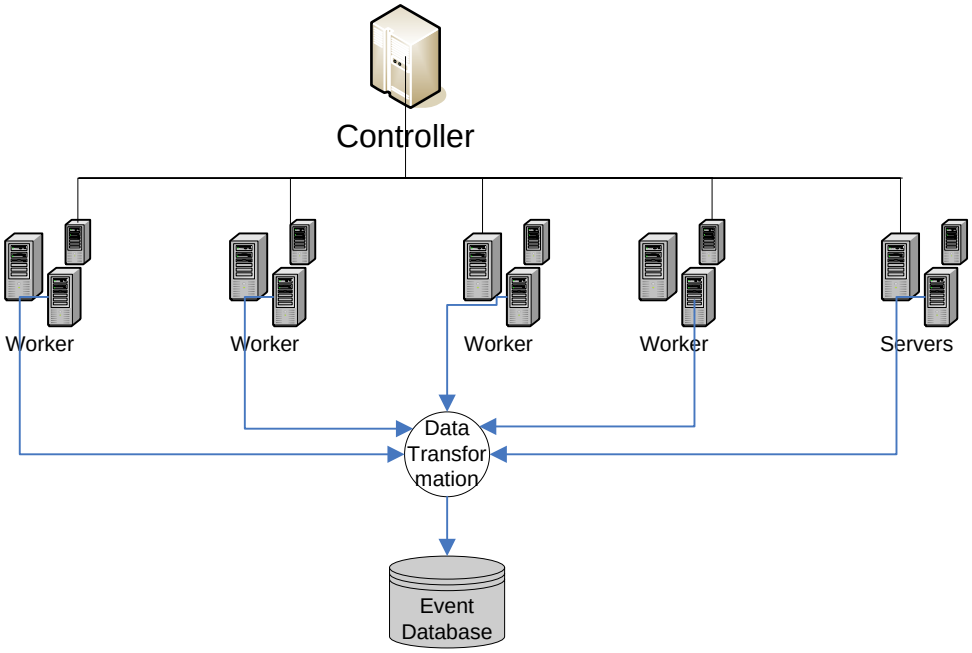


Figure 5: Automated Replication System

The replication system is load balanced by a controller which dispatches work to the workers registered to the system. Each worker is equipped with two virtual machines, acting as client and server. The server VM runs goat servers (e.g. IRC, web, ftp, etc.) via a virtual network. The client VM is booted with a kernel mode monitor agent, prior to the execution of the target file. The monitor agent intercepts and monitors all system calls in kernel mode. Specifically, the monitor agent

- filters processes and events prior to the execution of the target file.
- monitors events on a per process basis.
- tracks and correlates object state (e.g. file handle, registry handle, etc.).
- intercepts system calls with parameter information.
- captures subsequent status of system calls.
- monitors remote thread injection and process creation, for process and thread spawning.
- monitors network traffic.

All events in the system are captured and recorded to a log file. Event data is later transformed into a canonical format and stored, along with meta-data such as detection, family, hash, date, etc. The data is then fed to the clustering and classification processes.

Experiment Limitations

Our experiment implementation has several limitations at the time of completing this paper. We are working to resolve them in later experiments.

- The similarity measure ignores inter-process/thread relationships.

- The similarity measure does not fully exploit semantics in event objects. Specifically, the cost matrix we define only takes into account of basic event types (e.g. registry write, file read, etc.).
- Clustering time and space complexities are high. Our current implementation takes 3 hours to cluster 400 files with 1000 events limit (file can often have some where between 10000 and 20000 events). Event data occupies storage on average 150 malware/GB. We are working on optimizing both run time and data storage solutions.
- Some malware do not run properly in VM. Some packers are known to detect VM environment and prevent the file from normal execution.
- Runtime environment simulation is still primitive. For example, we have not implemented Instant Messaging or P2P applications/servers.
- We have not yet implemented any type of event aggregation, for example, multiple continuous writes a socket may be considered one single operation.

Data Analysis

For test data sets, we selected, in two separate experiments, 461 samples of 3 families and 760 samples of 11 families.

	Families	Number of Samples
Experiment A	Berbew	218
	Korgo	133
	Kelvir	110
Experiment B	Berbew	218
	Korgo	133
	Kelvir	110
	HackDef	88
	Bofra	7
	Bobax	159
	Bagz	31
	Bropia	7
	Lesbot	2
	Webber	1
	Esbot	4

Table 6: Data Sets in Experiments A and B

To evaluate our methodology, we ran 10 fold cross validation on the above two datasets. For each fold (iteration), we randomly pick 90% of data for training and keep the other 10% of data for testing. Error rate is calculated by dividing the number of variants incorrectly classified by the total number of variants for testing. This iteration is repeated 20 times. We vary and contrast experiments by adjusting two parameters:

- number of clusters, K
- maximum number of events, E

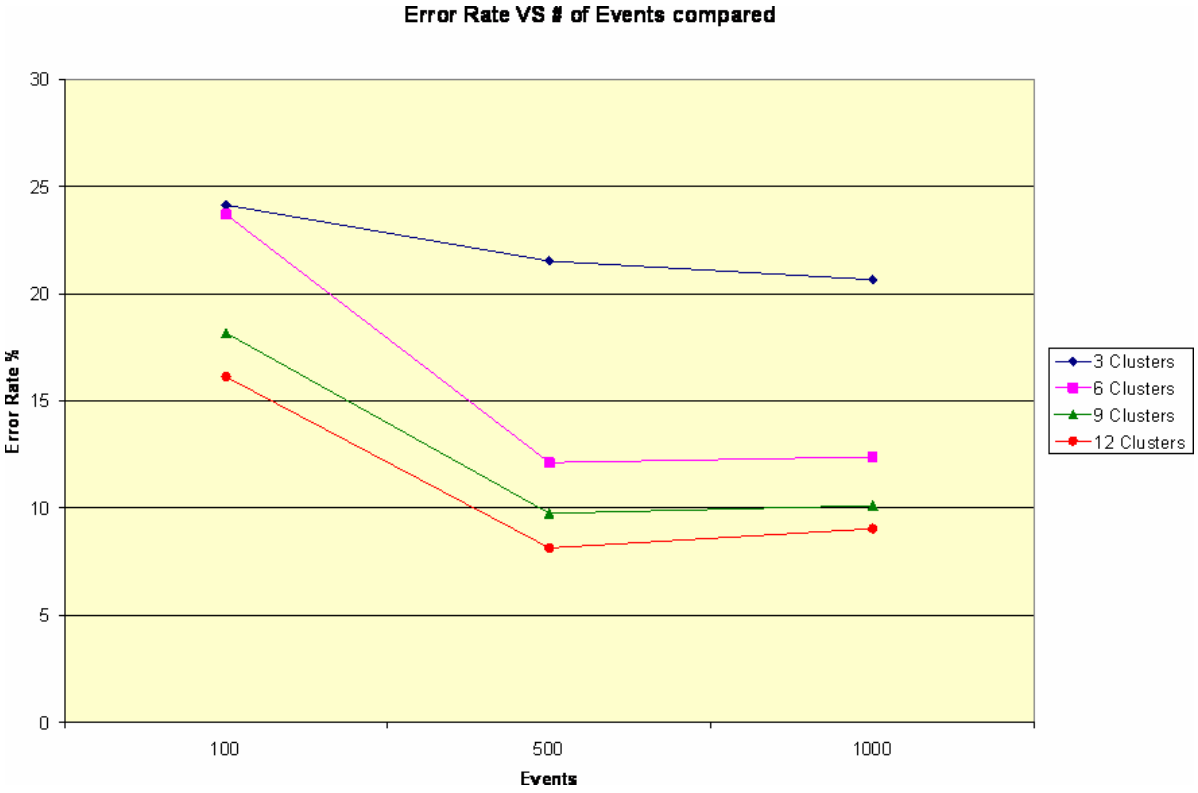
K is set to the multiples of the number of families in the dataset (m) from 1 to 4 (i.e. k = m, 2*m, 3*m and 4*m). E is set to 100, 500 and 1000.

We measure two metrics for evaluations:

1. Error rate is defined as
   ER = number of incorrectly classified samples / total number of samples. Accuracy is defined as
   AC = 1 – ER
2. Accuracy Gain of x over y is defined as
   G(x,y) = | (ER(y) – ER(x))/ER(x) |

#Events /#Clusters	100	500		1000	
	Error rate (%)	Error rate (%)	Gain over 100 events	Error rate (%)	Gain over 100 events
3	24.13	21.52	0.108164	20.65	0.144219
6	23.7	12.13	0.488186	12.39	0.477215
9	18.17	9.76	0.462851	10.12	0.443038
12	16.12	8.15	0.494417	9.05	0.438586

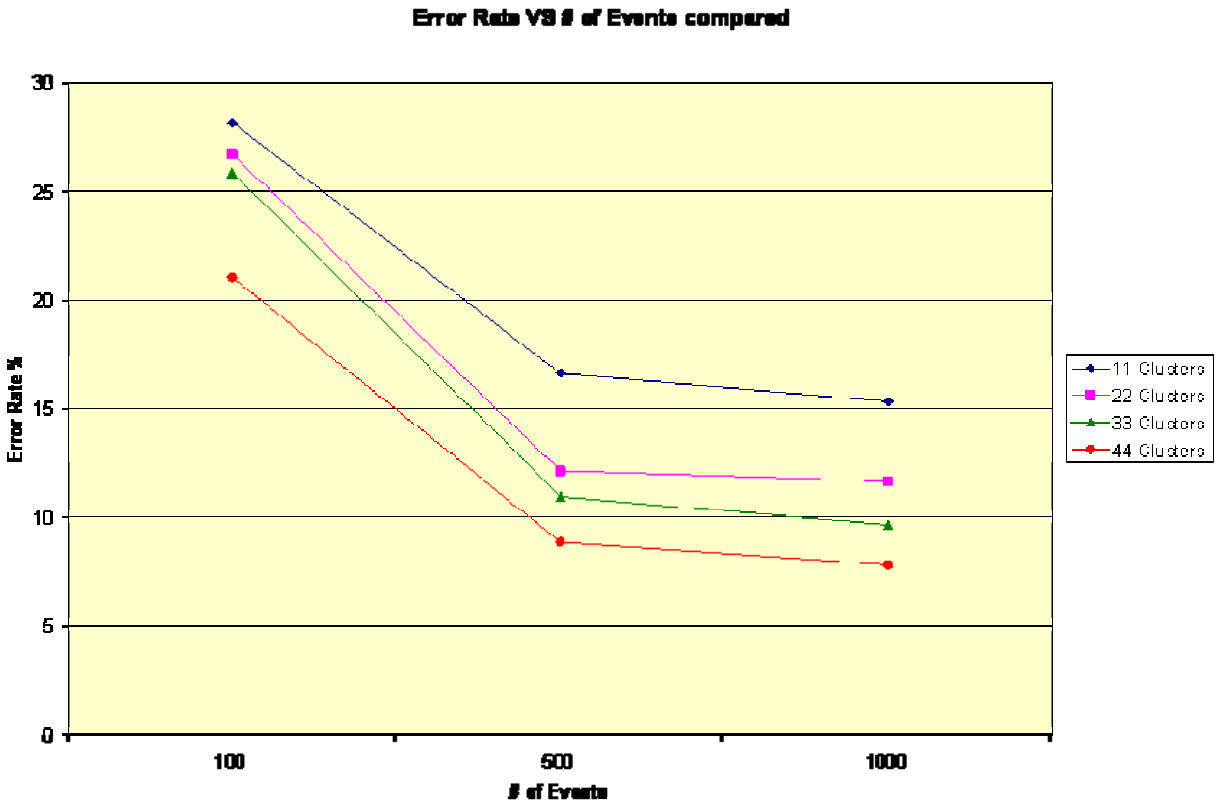
Table 7: Data from Experiment A



Graph 1: Error rate vs. Number of Events, by Number of Clusters, Experiment A

#Events /#Clusters	100	500		1000	
	Error rate (%)	Error rate (%)	Gain over 100 events	Error rate (%)	Gain over 100 events
11	28.17	16.63	-0.409656	15.32	-0.456159
22	26.72	12.14	-0.545659	11.67	-0.563249
33	25.87	10.98	-0.57557	9.65	-0.626981
44	21.06	8.87	-0.578822	7.84	-0.62773

Table 8: Data from Experiment B



Graph 2: Error rate vs. Number of Events, by Number of Clusters, Experiment B

Key Observations

- Accuracy vs. #Clusters
  Error rate reduces as number of clusters increase. When the number of clusters increases, the purity (accuracy) of each cluster increases, which in turn gives us better cluster representatives, subsequently improving the classification accuracy.
- Accuracy vs. Maximum #Events
  Error rate reduces as the event cap increases, because the more events we observe, the more accurately we can capture the behavior of the malware, hence more likely the clustering discovers the semantic similarity among variants of a family. However, the increase in number of events also compounds the effects of outliers (skewed data points). This side effect can be observed from the trend in graph 1, where increase in events from 500 to 1000 caused a slight increase in error. When we drilled down into the set of experiments, we observed that we were getting a few clusters of 1 to 2 samples (outliers), and since K is fixed, this results in more samples classified into fewer clusters, therefore more errors. The reason for the increase in effect of outliers is from the fact that the outliers simply have much longer event sequence than the rest, which was not a problem when we had lower event number cap.
- Accuracy Gain vs. Number of Events
  The gain in accuracy is more substantial at lower event caps (100 vs. 500) than at higher event caps (500 vs. 1000), which indicates that between 100 to 500 events, the clustering had most of the information it needs to form good quality clusters.

- Accuracy vs. Number of Families
  The 11-family experiment outperforms in accuracy the 3-family experiment in high event cap tests (1000), but the result is opposite in lower event cap tests (100). As we investigate further, we found that the same outliers were found in both experiments, and because there were more semantic clusters (11 vs. 3), the outlier effects were contained.

CONCLUSIONS

As malware continue to trend toward multi-vector infection, diverse payloads, prevalence of spyware/adware, increasing inter-component and system interactivity, we see considerable challenges to existing classification methodologies. The analysis of run time behavior coupled with machine learning allow us focus on pattern/similarity recognitions in behavior semantics, rather than merely binary structures, across large number of files, in an automated and adaptable manner. Limitations such as lack of code structural information, environment conditions, and ineffective emulation/virtualization also suggest the importance of combining static analysis to improve classification accuracy. Developing automated classification process that applies classifiers with innate learning ability on near lossless knowledge representation is the key to the future of malware classification and defense.

References

[1] Jeff Kephart, Dave Chess and Steve White (1997). Blueprint for a Computer Immune System. IBM, Proceedings of VB Conference.

[2] Ford R.A., Thompson H.H. (2004). The future of Proactive Virus Detection. Proceedings of EICAR Conference, Luxembourg.

[3] Wagner M. (2004). Behavior Oriented Detection of Malicious Code at Run-time. M.Sc. Thesis, Florida Institute of Technology

[4] Richard Ford, Jason Michalske (2004). Gatekeeper II: New approaches to Generic Virus Prevention. Florida Institute of Technology.

[5] Mazeroff G., De Cerqueira V., Gregor J. and Thomason, M. Probabilistic Trees and Automata for Application Behavior Modelling.

[6] Matthew M. Williamson (2005). Using Behavior to Detect and Classify Information-Stealing Malware. Sana Security, Inc. USA.

[7] John Canavan (2005). The Evolution of Malicious IRC Bot. Symantec, Proceedings of VB Conference.

[8] Marius Gheorghescu (2005). An Automated Virus Classification System. Microsoft Corp, Proceedings of VB Conference.

[9] Sabin. T. (2004). Comparing Binaries With Graph Isomorphism. http://razor.bindview.com/publish/papers/comparingi-binaries.html

[10] Dullien.T, Rolles. R. (2004). Graph-based Comparison of Executable Objects. http://www.sabre-security.com/files/bindiffsstic05.pef

[11] L. Kaufman, and P. J. Rousseeuw (1990). Finding groups in data: An introduction to cluster analysis. New York: John Wiley & Sons.

[12] J. MacQueen. Some methods for classification and analysis of multivariate observations. In Proc. 5th Berkeley Symp. Math. Statist, Prob., 1:281–297, 1967.

[13] K. Beyer, J. Goldstein, R. Ramakhrisnan, and U. Shaft. When is 'nearest neighbor' meaningful? In Proc. 7th Int. Conf. on Database Theory (ICDT'99), pages 217–235, 1999.

[14] F. Farnstrom, J. Lewis, and C. Elkan(2000). Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2: 51–57.

[15] Boriana L. Milenova, and Marcos M. Campos. O-Cluster (2002). Scalable Clustering of Large High Dimensional Data Sets. *International Conference of Data Mining* (*ICDM),* 290-297 http://www.oracle.com/technology/products/bi/odm/pdf/o_cluster_algorithm.pdf

Enlisting the End-User – Education as a Defense Strategy

Jeannette Jarvis
Boeing, USA

Abstract

Protecting an enterprise from computer malware requires a multi-tiered security product and process approach. This defense in depth strategy is a good policy to provide for a robust and secure environment. However one countermeasure that is often overlooked is end-user education. The end-user is the last line of defense for any threat. Are they going to do the right thing? Do they know what to do? Can they be effective?

As our environment becomes more virtual and the end-user is mobile most of the time, where they aren't protected from the strong perimeter protection your enterprise may provide, it is imperative that they know the part they play in keeping malware threats at bay and that they make the correct choices.

Evolution from a Honeypot to a distributed honey net

Oliver Auerbach
Avira GmbH

About Author

Oliver Auerbach is a virus researcher at Avira GmbH since ’99. Besides he is studying informatics at the University of Applied Sciences in Weingarten, Germany.

Contact Details: Avira GmbH, Lindauer Str. 21, D-88069 Tettnang, Germany, phone +49 (0) 7542 500-0, fax +49 (0) 7542 525-10

Keywords

Honeypot, proactive, exploit, detection, network infection, string matching, attacks, infection simulation

Evolution from a Honeypot to a distributed honey net

Abstract

Over the last few years worms and bots in particular became a penetrating widespread threat. Antivirus companies have developed better and better heuristic detection routines against these pests but some are still slipping through. Therefore, the old method of adding signatures as soon as a new variant shows up is still very important. At a time when more than 40 new bot variants appear each day, it is extremely important to have a binary sample for in house analysis.

A "traditional" Honeypot that captures the latest variants is a quite efficient technique. A slight disadvantage is that most of the attacks are from the same subnet and after a while, you are aware of all bots around you and less new variants are discovered. For increased efficiency, you have to set up more and more honey pots in different locations, especially different subnets. Usually this leads to a large amount of administrative effort, which implies fine-tuning on the honey pots behaviour each time a new technique or exploit is discovered to infect a machine.

This article describes how we managed to extend our Honeypot, which was especially designed to capture worms, to a honey net that solves this problem using a new technique. All information about new attacks and samples are collected at a single point. Further configuration and changes can be made from only one point and all this is possible in real time.

Introduction

Using a Honeypot to increase the level of security inside your own network is everything but new. Nowadays there are lots of Honeypots available and security experts make use of this technology in various approaches. While the majority is used to keep hackers away from production machines some of them are used to capture malware samples. Mwcollect or the open source project Nepenthes are good examples of public available Honeypots focused on this field of interest.

Lately it became extremely important for the antivirus industry to be proactive instead of relying on customer submissions. Therefore most antivirus vendors use either public available Honeypots, like those mentioned above, or others that are made in-house. The most primitive but very effective Honeypot technology is seeding of email addresses on the internet and monitoring their related accounts. Everyone knows that that a huge number of infections doesn't take place by email anymore but more than ever via direct network infection. The more specific targets are vulnerable Windows operating systems.

Honeypot technology comes in very handy to cover the last mentioned target of attacks. Due to this we wrote our own in-house made clone and we continuously improved it in the way we considered as being most effective.

The paper shows the results and the development evolution including, future integration and ideas such as: requirements of a stand alone pot, forwarder technology, protocol implementation, packet database, infection simulation, etc.

Requirements:

The ideal trap from our point of view is the one that is not limited to threads coming from a certain operating system or service which relies on a specific protocol. You never know what the next kind

of attack looks like so the Honeypot should be extensible. Practically this means that a trap operating on a Windows system should have the capability to catch worms designed for a Linux operating system, just to have one example mentioned. Besides that we have to pay attention to fingerprinting [1] though it is not yet a common technique used by malware. However it's not only fingerprinting but any kind of method that leaves room for revealing the trap.

According to those requirements the Avira pot was developed for windows and uses the packet filtering framework WinpkFilter [2] which allows full control on raw network packets.

Stand alone Honeypot:

The first version was able to simulate the vulnerability MS04-011[3] that was used a lot by Korgo at that time and the famous DCOM RPC vulnerability MS03-026[4] that was first used by Lovsan - alias Blaster. After a very successful start with sometimes a dozen of new variants a day, less and less new malware was discovered. Finally, we started to implement more vulnerability behaviour as well as support for some common commands from the SMB protocol that were used by worms in order to propagate.

Each new implementation brought another peek of new variants but always turned back to the previous, low activity stage. An analysis of the source IP addresses pointed out that around 80 percent of the infections originated from the same subnet. At that time, the Honeypot was located in the "Deutsche Telekom AG" subnet and the IP address was 80.133.x.x. This ISP is the largest in Germany and they provide connectivity to the Internet for maximum 24 hours. After max 24 hours the connection is terminated and the user has to dial-up again. Consistently, they provide a new IP address each time you dial up. Unfortunately, we found ourselves always in 80.133.x.x, even if we reconnected dozens of times. We were sure that other not-yet-seen variants were active in other subnets.

Other not-yet-seen variants are active in other subnets as the worms usually try addresses similar to their own, in order to have more success. These addresses receive more hits than the random generated ones, not to mention their validity [5]. Finally we set up various stand-alone Honeypots and (you might have guessed it already), the distribution of the traps worked pretty well and we got new variants of worms that we hadn't ever seen before. Every time a new trap was put online, we were able to see another peak.

A mailing routine was added to collect all the samples on a central server. The binary was shipped along with the original IP and port, timestamp, original filename and the whole infection log, which could help to determine bugs in the program as well as transmission errors or ideas for further implementation. At the time the email arrived on the server, a couple of error checks were made, and it was decided if this had to be forwarded to the lab for further analysis and statistics purposes.

The Honeypot farm raised other problems as we had to maintain more than one trap, update it, collect and interpret the results. Moreover, we only had data from successful infection processes but no data about new techniques to compromise a machine. Those drawbacks forced a redesign of the whole concept which practically made the change from one or more stand-alone Honeypots into a distributed honey net.

Forwarder technology:

The project was split into two parts. One is called “Forwarder” and the other - “MainPot”.

As you may imply, the “Forwarder” will redirect the traffic to a certain IP address and port. The traffic arriving on certain ports at the forwarder side is not just a simple redirect using NAT . As we want to create reliable statistics, watch out for seeding attempts of new malware and figure out in which subnets certain malware is active, an own layer (*Figure 3*) on top of the TCP protocol was implemented.

The new protocol contains data such as identifier, original IP, original source and destination port, a number from config.cfg file and some free Bytes reserved for further ideas. The identifier helps the “MainPot” to distinguish if the packet really comes from a “Forwarder”, if it was just a port scan or any other data arriving on that port. The number set in the configuration file could be any from 1 to 65535 and uses two Bytes in the protocol. It has to be set manually during the “Forwarders” first installation process and helps to determine where the forwarder is physically placed. It is necessary to ship data such as the original source and destination port along, as the “MainPot” has to know how to react when constructing the answer packet. The original IP address is shipped in order to provide statistics.

However, the “MainPot” reconstructs the original packet which initially arrived on the “Forwarder” side. Afterwards it searches its database for the type of data that the answer packet should contain. If it was successful, which means that this was already implemented it constructs the answer packet. Finally, the new protocol header is stripped up again and sent to the “Forwarder”. This has the simple task to replace the original IP and the ports, recalculate the checksums before sending the answer packet to the attacker. Depending on the connection and location, the delay between the “Forwarder” to the “MainPot” is around 200ms. The time both Honeypot parts need to construct the packet is less than 10ms, which means that most of the time is spent on the road but it is still fast enough for our needs.

After we implemented more and more forwarders we had to face the problem that some routers didn’t like our data packets. The solution was to transform the raw data into UDP and later to reconstruct the original TCP on the “MainPot” side.

Protocol implementation:

Most of the attacks we’re facing are for Windows computers on port 135, 139 and 445. Therefore they mainly rely on the RPC and SMB protocol.

However there is no actual need to implement the whole capability of the protocols and their dozen of commands in the Honeypot itself. Today’s malware uses only very few commands in order to break into vulnerable systems. Due to the limited amount of communication it would be enough even to hardcode the answer packets in the Honeypot itself. Exactly this was done in one by the early versions of this Honeypot and we were still able to capture 142 different working samples within the first three weeks.

Nevertheless the goal is to be flexible when it comes to new vulnerabilities that are exploited by network worms. The main interest is to extend the Honeypot in a fast and easy way so that it would be capable to catch the latest threats. This led us to some sort of database implementation that can be maintained separately.

Packet database:

Except the TCP three way handshake and some other minor parts the Honeypot works with a sorted list of data records stored in memory. For every data packet a CRC32 is calculated and stored with some other information such as TCP port, packet length, comment, reference CRC and not to forget the raw data itself.

- Counter: How many times the packet was intercepted in total
- TCP port: Needed to simulate a recorded infection against a vulnerable machine
- Packet length: length in Bytes of the raw data
- CRC: CRC32 from intercepted raw data packet
- REF-CRC: Reference CRC which is practically the reference to the answer packet
- Comment: Servers for a human readable output

A short selection of packets out of the whole DB is shown below:

```
NO   CNT    PRT    LEN    CRC        REF-CRC    Comment:

0    907    139    133    1C23675C   470913A9   NetBIOS - SMB - Negotiate Protocol Request
1    0      135    60     41E304A4   00         RPC Positive Bind Response
2    0      139    89     470913A9   00         NetBIOS - SMB - Negotiate Protocol Response
3    0      139    4      47507EB0   00         NetBIOS - Positive session response
4    130    135    104    78E5220A   41E304A4   RPC Bind Request
5    841    139    72     E7940B20   47507EB0   NetBIOS - Session Request to SMBSERVER


s         Show database content
d         Delete entry
f         Write buffer to file
c         Set or update comment
r         Set or update R-CRC
x         Export existing entries to a new DB file
q         Quit

Please choose...:
```

Some of the data packets don't need to be answered with other real data but with a confirmation packet. This is implemented hard coded and controllable via a REF-CRC "01" which has to be set manually.

All new and not yet known packets are inserted by default in the DB. The Honeypot does not know how to reply to those packets and some manual work is required here. In order to teach the Honeypot with answer packets and finally the behaviour as a vulnerable system a Trace.log file is created for each infection. It contains the CRC, REF-CRC, sequence number and acknowledge number of the TCP protocol.

Infection simulation:

Using the information logged in Trace.log and described above we're able to simulate the whole communication against a vulnerable system. In our case the first choice is always a Windows XP

machine without any service pack installed. For a better understanding in the following section a sample of the trace.log file is shown below:

```
E7940B20,47507EB0,88CBDFC8,C5CC193D
9DD277AF,01,88CBE010,C5CC1941
1C23675C,470913A9,88CBE014,C5CC1941
111CBC4C,01,88CBE099,C5CC199A
E88C9DF9,00,88CBE641,C5CC199A
```

a) The first unknown packet is simulated against a vulnerable XP machine and the answer packet stored in the DB

b) Next time the same network worm attacks it will face the answer of the vulnerable XP machine as this was already stored in the DB and the REF-CRC was registered.

c) It's likely that also this packet is unknown and stored in the DB for further manual procession. Step a) is repeated and also the answer of packet two is stored in the DB.

d) This process is continued until we reach the exploit code which most likely contains any code in order to open a remote shell and execute a command. We're searching explicitly for this kind of command, parse it and finally download a binary sample of the malware itself.

Infection log:

In order to see how the infection takes place the following infection log is created. It serves our researchers as a source of information about how to deal with this threat.

```
  IP-Address      From  To    Info:

I 81.181.164.247  3424  139   SYN
O 192.168.0.2     139   3424  SYN + ACK
I 81.181.164.247  3424  139   ACK (No data)
I 81.181.164.247  3424  139   ACK + FIN
O 192.168.0.2     139   3424  ACK + FIN

I 81.181.164.247  3436  139   SYN
O 192.168.0.2     139   3436  SYN + ACK
I 81.181.164.247  3424  139   ACK (No data)
I 81.181.164.247  3436  139   ACK (No data)
I 81.181.164.247  3436  139   ACK + PSH
                              Search DB: e7940b20 (NetBIOS - Session Request to SMBSERVER)
                              MATCH
                              Search DB for R-CRC: 47507eb0
                              MATCH
O 192.168.0.2     139   3436  NetBIOS - Positive session response
I 81.181.164.247  3436  139   ACK + PSH (Answer to existing SEQ number)
                              NetBIOS Session message: 133 Bytes
O 192.168.0.2     139   3436  ACK only
I 81.181.164.247  3436  139   ACK + PSH (Answer to existing SEQ number)
                              Search DB: 1c23675c (NetBIOS - SMB - Negotiate Protocol Request)
                              MATCH
                              Search DB for R-CRC: 470913a9
                              MATCH
O 192.168.0.2     139   3436  NetBIOS - SMB - Negotiate Protocol Response
I 81.181.164.247  3436  139   ACK + PSH (Answer to existing SEQ number)
                              NetBIOS Session message: 4287 Bytes
O 192.168.0.2     139   3436  ACK only
I 81.181.164.247  3436  139   ACK + PSH (Answer to existing SEQ number)
                              CMD command found!
```

Please note that the “CMD command found!” string indicates a match when searching the exploit code for several commands. A sample string may look like:

cmd /k echo open 81.181.164.247 30144 > o&echo user 1 1 >> o &echo get winfixup.exe >> o &echo quit >> o &ftp -n -s:o &del /f /q o &winfixup.exe

It simply creates a file called “o” with instructions used by the ftp program in order to download the malicious file and execute it afterwards. This is a common method used by various network worms but especially bots.

References

[1] Nmap: http://www.insecure.org/nmap/

[2] WinpkFilter: http://www.ntkernel.com/

[3] Microsoft Security Bulletin MS04-011:
http://www.microsoft.com/technet/security/bulletin/MS04-011.mspx

[4] Microsoft Security Bulletin MS03-026:
http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx

[5] Gabor Szappanos, 'Advanced survival techniques in recent Internet worms' Proc. Virus Bull. Int. Conf., 2004 (abstract: http://www.virusbtn.com/conference/vb2004/abstracts/gszappanos.xml)

Fighting against Phishing for On-Line Banking Recommendations and Solutions

Marko Helenius
University of Tampere

About Author

Marko Helenius is Assistant Professor at the Department of Computer Sciences, University of Tampere. He is also the coordinator of the Finnish Information Security Consortium and a board member of the Finnish Society for Computer Science. He graduated from the University of Tampere in 1994 and his post-graduate research resulted in high quality antivirus products analyses. He received Ph.D. in 2002 and his doctoral thesis presented a system for automating several tasks associated with assessing antivirus products' virus detection capabilities. Nowadays, his research and teaching concentrates on malware prevention.

Contact Details: Department of Computer Sciences, Kanslerinrinne 1, 33014 University of Tampere, Finland, phone +358-3-3551-7139, fax +358-3-3551-6070, e-mail marko.helenius@cs.uta.fi

Keywords

phishing, on-line banking, pharming, fraud, malware, virus, worm, information security

Fighting against Phishing for On-Line Banking Recommendations and Solutions

Abstract

Phishing against on-line banking has become a serious problem in the cyber world. I will discuss some of the reasons for the situation and propose some solutions and recommendations that will help in the fight against phishing. I will take three different perspectives: consumer's perspective, service provider's perspective and regulation authority's perspective. A classification of phishing attacks will be presented. For illustration also a local example of a phishing case will be discussed.

Introduction

Phishing has quickly become a world-wide problem. International dimension of the e-commerce has made phishing appealing. Malware is being used for financial gain and by organised crime. Criminals are able to do successful frauds and therefore they have resources for co-work and sophisticated attacks. While phishing has become a real problem there is so far only little preventive research in the area. The aim of this paper is to contribute to the prevention of phishing and propose methods that can be used against phishing. Furthermore, raising awareness is also a major objective of this paper.

There seems to be little research in the area of phishing that has been published so far. However, practical work is being taken seriously. For example, Anti-Phishing Working Group (2005) is doing valuable practical work against phishing. Furthermore, authorities are doing valuable work in prevention of phishing.

Phishing against on-line banking is the main focus of this paper. Typical credit card fraud has been delimited outside the scope of this paper. However, other services associated with on-line banking (e.g. loan services, stock market and other financial services) are in the scope of the paper. In this paper we have the following research question:

- What principles can be found that could help restraining phishing against on-line banking?

The results of this work are obtained from the source material and experiential knowledge gathered during my research work at the University of Tampere. In addition, students' comments during a course "malicious programs and their prevention" were used.

We will next outline the phishing concept and propose a classification of phishing attack methods. We will then continue proposing possible solutions and recommendations restricting the phishing problem.

Reasons for the phishing problem

We will at first briefly discuss some of the reasons for the phishing problem. At first we must notice that phishers may use sophisticated techniques and there is often organized crime and significant financial gain involved. As Emigh (2005, p. 6) observes:

"*It is a common misconception that phishers are amateurs. This is not the case for the most dangerous phishing attacks, which are carried out as professional organized crime. As financial institutions have increased their online presence, the economic value of compromising account information has increased dramatically. Criminals such as phishers can afford an investment in technology commensurate with the illegal benefits gained by their crimes.*"

For criminals phishing is an appealing method of crime, because the chances to be caught are low. Furthermore, international dimension protects criminals. The victim may be in the other side of the world while criminals may efficiently hide themselves. Furthermore, in some countries there may not be preventive legislation or the legislation is simply ignored, for example, because of corruption.

Because on-line services need to be public, an attacker can easily copy all the content available from real services. Therefore it is difficult to distinguish fraud web sites from real services. Futhermore, it is in a criminals' interest to make user requests to fraud web services as convincing as possible.

The insecure while still cost-efficient Internet infrastructure is being used for financial services. A service provider as well as a user may choose convenience over security. This is appealing, because convenience of the service usage is likely to affect usage of the on-line banking service. Still customers may have little possibilities for affecting security of the services. Meanwhile, more and more users are using e-commerce services and banks are encouraging the usage of on-line services. Moreover, insecure authentication is used for transactions, which may be based on a user ID and a password. Thus there is no way to authenticate the actual user.

Users of the Internet and online services include all levels of expertise and capabilities. While some users will learn secure computing, there will always appear new inexperienced users. From a customer's point of view there is no certainty that the hyperlinks in e-mails or web pages are correct. Furthermore, links are easy to click and even a single click may compromise a computer.

Catching up the criminals requires extensive resources and international authoritative co-operation while for criminals the attacks are easy and cheap to implement. With basic tools a successful attack may be achieved. Furthermore legislation does not typically take in account phishing. Phishing is assessed as a minor crime compared to physical theft. From a criminal's point of deceiving few victims may be sufficient. One element is that criminals do not need to know victims in person. Therefore conscience may not prevent a fraud.

Finally, I would like to add that there is lack of security in hardware and network architectures. In other words there is lack of security in fundamental parts of information society. Typical devices and network protocols were not designed for such sensitive usage. Typical computer architectures and network protocols are by their nature insecure. Yet they are being used for commercial services over insecure connections.

Classification of phishing attack methods

I will next propose a classification for phishing attacks.

User request

User request include all the requests made by a phisher. Probably the simplest method is to ask information from a customer. Typically e-mail is used for misleading a user to a fraud web site (see Appendix 1 which contains definitions of some terms) However, fraudulent user requests may be accomplished by other ways, too: using a web page, telephone, VoIP (Voice over Internet Protocol), snail mail etc. A typical user-request is placed on a fraud web page that deceives authentication information from a customer. Lance (2005, p. 160) describes also a reply attack which is based on an attack performed during a valid advertising campaign.

Typical for user-request is that they utilise social engineering techniques. Typical user requests include, e.g. alleged problem with a user account, new services being introduced, security improvement, participation to an anti-fraud program, consumer surveys, fake rewards and false invoices (see phishing archive, Anti-Phishing Working Group 2006).

Redirection

Redirection means the techniques used for directing a user to a fraud web site. Such techniques include DNS-redirection, web page redirection (Lance 2005, p. 152), search engine manipulation (see Muttik 2005) and e-mail link redirection.

Muttik (2005) discusses also DNS-poisoning (Domain Name Service), also called pharming. In DNS-poisoning desired DNS-addresses are redirected to fake addresses. As Muttik (2005, p. 23) observes "*There are several ways to introduce a malicious DNS modification – exploits in DNS protocol, hacking into a DNS server, social engineering*".

Tocheva and Rautiainen (2005) describe different techniques used in e-mail links for misleading users to fraud web sites. A criminal may use IP address instead of DNS name, the real link may be different from the link displayed, percent or Unicode escape characters may be used, encoded HTML entities may be used, username/password part of the link may be misused and scripts may be used to hide links.

User-end-hijacking

The most alarming technique is the possibility to hijack end-users' systems. Typically this is achieved by Trojan horses, vulnerabilities or viruses (see Appendix 1). Once the system has been compromised an attacker can do whatever preferred. The victim may be connected to a fraud web site and see everything like connected to a real service. This can be achieved by using so called rootkit techniques (see, for example, Hoglund and Butler 2005; Kasslin et al. 2005). User-end-hijacking is appealing from a criminal's point of view, because it is likely that there are enough consumers who do not have good enough security and therefore there are computers to compromise.

Man-in-the-middle attack

In the man-in-the-middle attack the attacker is between the real service and the victim. When the victim sends information to the fraud site, it may be selectively passed to the real service and the responses may be fake or valid responses from the real service. Once an attacker is able to achieve sophisticated man-in-the-middle-attack

If well enough implemented, man-in-the-middle-attack is difficult to prevent. As Schneier (2005) observes, man-in-the-middle attack may be successful even against two-factor authentication and disposable passwords (see Appendix 1).

Banking–side-hijacking

Theoretically it is possible to hijack the banking side. Although this seems to be currently a minor threat and is not therefore discussed in detail, an attack against banking side should also be considered as a threat. Fortunately, banks have the motivation to put into practise all the necessary known security measures. However, sometimes these security measures may not be enough and an attacker may be able to compromise a banking side.

Relationships of attack methods

It is important to notice that theoretically all these methods may be combined. Furthermore, theoretically with certain conditions user requests, user-end-hijacking, man-in-the-middle-attacks and banking-side-hijacking can be used as a single phishing method. We will next discuss some of the typical combinations.

A user request is (e.g. deceiving a user to perform some security action) can be followed by a redirection (e.g. a link to a phishing site in an e-mail). After that a new user request is made (e.g. by using fraudulent forms in a phishing site) and thus the banking site authentication information is deceived from the victim. An alternative for a user request is a man-in-the-middle attack (Figure 1).

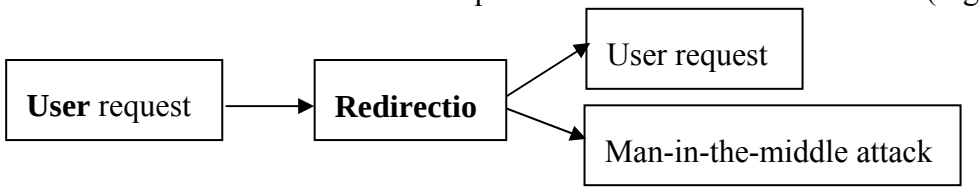


Figure 1: Attack pattern 1

In a more sophisticated attack the user-end is at first hijacked. After that a user is directed to a phishing site (e.g. using a rootkit when a user aims to visit a real service). After that the attacker transmits information from the real service and performs a man-in-the-middle attack or a user request is made (Figure 2).

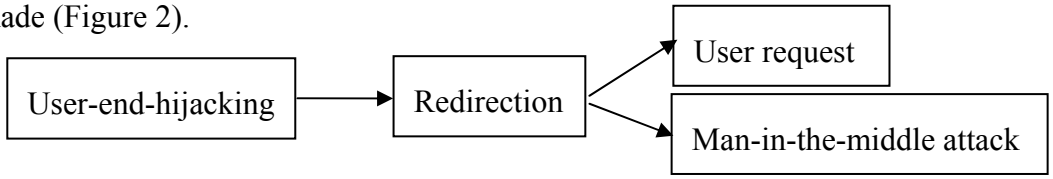


Figure 2: Attack pattern 2

In case of a search engine attack a user is directly redirected to a phishing site. Again this can be combined with a man-in-the-middle attack or a user-request (Figure 3).

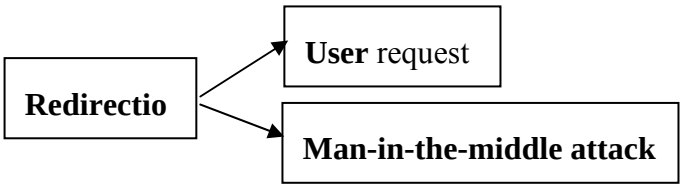


Figure 3: Attack pattern 3

In case of DNS-poisoning attack there must be at first a man-in-the-middle-attack where a user is redirected to a fraud web site. Again this can be combined with a man-in-the-middle attack or a user-request (Figure 4).

Figure 4: Attack pattern 4

From Figures 1-4 we can observe a typical pattern: a redirection is followed either by a user request or a man-in-the-middle-attack.

Solutions and recommendations

We will next discuss possible solutions to restrain the phishing problem. At first we must observe that there are no single solutions and there need to be both social and technical solutions. We will notice that there are solutions that could be easily accomplished. Yet still there are solutions that are not extensively in use. Three different perspectives are proposed: customer, banking, and authorities.

Consumer perspective

Consumers seem to be the weakest link, because among the possibly millions of consumers there will be weakly protected. Yet most sophisticated attacks can be so well prepared that even professional computer users may be cheated. However, consumers can do plenty of things in a fight against phishing. In the following are some suggestions what consumers can do.

Right for security

Demand security from on-line banking. After all, security should be the right of the consumer. If security of one service is not good enough, try another one.

Protect your computer

Ensure that the computer used for on-line banking is protected as well as possible. Although all known methods of protection are vulnerable they will decrease the possibility for user-end-hijacking. Especially when different protection methods, like firewall protection, anti-virus protection, spam filtering, anti-spyware protection, security updates, hardening and diversity are combined the possibility of user-end-hijacking decreases accordingly.

Ensure correct service

Do your best to ensure that the financial service web page is a correct one. Ensure that the address is shown correctly and examine the certificate of the web page. There exist programs that try to protect from end-user-hijacking and DNS-poisoning attacks. Furthermore, there exist secure toolbar programs that aim to show suspicious and trusted web pages.

Do not trust e-mail

Do not follow links in e-mail messages. If a bank sends an e-mail regarding user accounts, assume that the e-mail is a phishing attempt. Banks have a policy (if not should have) that e-mails requesting user account information is not sent. Even if the e-mail itself is convincing, remember that it is in the criminals' motive to be as convincing as possible and therefore criminals utilise convincing social engineering techniques. If you are, nevertheless, in doubt contact your bank, but never fill in sensitive information following an e-mail.

If possible, use secure e-mail programs. For example, with certain Linux programs it is possible to read html-messages in plain text and attachments must be saved before they can be used. Make e-mail program usage as clear and secure as possible.

Demand restrictions to on-line banking

As we know it is not possible to obtain perfect self-protection. Be aware of what is possible by using the on-line account and what is your responsibility. Is it possible to apply for loans and are the loans admitted without personal contact? Is it possible to restrict the amount of bank account usage? If restrictions are not possible, it is possible to use on-line banking services in other bank than the one used for other financial services. Main financial fortune can be kept safe.

Be aware of agreements

Read carefully all the agreements of on-line banking. If you do not find them appropriate, do not sign the agreements (or cancel agreements already signed) and state the reason of disagreement. Do not trust appealing wording and convincing (e.g. phishing is a marginal problem and has never happened to us). If convincing is made verbally, it will not help in case of misuse.

Give feedback

If possible, give written feedback. Although one person may not affect bank policies if there are several, banks are likely to take in account more secure policies. However, even one well grounded and constructive feedback may have an effect.

Follow your bank account

In case of fraud it is essential to act as quickly as possible. Prepare also for identity theft. In case your social security number, address, phone, personal identities, bank account information etc., are stolen the possibility for identity theft is present.

Casecontact tips

Casescontact (2006) describes some practical tips for end-user protection. Rights of certain programs can be limited, for example by a special tool program (see Howard 2004). Digital certificates from banking services can be stored on a secure location, like off-line computer, memory stick etc. Besides, certificates should be encrypted. Password lists must be kept in a safe location. Password and username should not be typed, but pasted in order to make it harder for keyloggers (see Appendix 1) to obtain them. Shared computers should not be used for on-line banking. Hyperlinks in e-mails must not be clicked. Always log out from the service. The web browser must be set so that it will not store encrypted pages in the cache memory. Finally, one must be suspicious and remember that bank "*a) does not send you e-mails that contain a hyperlink connecting you to the logon page of your bank's online banking portal or a message that asks you to provide username and codes b) does not ask for sending via e-mail your credit card number, access codes, PIN or transaction codes.*"

Banking perspective

Banks are in a key position to prevent the problem. While banks are encouraging customers to on-line banking, they should also realise the serious responsibility. Furthermore, if customers are secured they are likely to obtain services. As Onkalo (2005) observes sense of security is one of the key elements in adaptation of e-commerce services. We will next discuss some solutions and recommendations from a banking perspective.

Confirmation about critical actions

It is possible to send confirmation about transfers and other critical actions, especially if they are suspicious. Although e-mail is cost-effective more security is achieved by using alternative communication channels (e.g. mobile phone networks). Furthermore, it may be possible to contact the customer in person in case of suspicious transactions.

If e-mail is used, it should be authenticated. Although properly implemented authentication improves security, it must also be realised that authentication may create false sense of security (Bellovin 2005, Ellison and Scheier 2000). Criminals may use fake authentication or once the victim's computer is hijacked they may display real looking authentication.

Personal restrictions

Allow customers to make personal limitations to on-line banking. It is possible to limit the amount of transfers within certain time, set delays to transfers, set limits to allowed actions (e.g. international transfers are not allowed, loan applications are not allowed, stock exchange is not allowed). Of course, personal restrictions must not be allowed to be loosened by using on-line accounts.

It may appear that limitations are restricting usage of on-line banking. However, if a customer does not use certain services, I cannot see any valid reason why restrictions should not be allowed. It is likely that possibilities for personal limitations will create trust on services as long as a customer knows his/hers preferences. For example, in Finland some banks have made possible personal restrictions, while others have not.

Delayed transfers

It is possible to delay transfers especially if they are suspicious. A customer will have time to make corrective actions. If a transaction is delayed, it would be possible to allow cancellation. Delays have already been used to restrict phishing (Layden 2005), but because of inconveniency delays are typically used only in extreme cases.

Agreements

Of course, legislation varies from country to country and therefore it is difficult to make extensive recommendations. It is however, in the interest of phishing prevention to make clear and understandable agreements that will benefit both financial service producers and customers. While each party is aware of liability it is easier to produce trustworthy services.

Authentication

In addition to user authentication also the financial service should be authenticated. Service authentication should be made before user authentication so that a user knows that the service is correct.

As a minimum recommendation there should be one-time password based user authentication. The confirmation password should be a one-time password. Not just for log in should be authenticated, but also each critical action should be authenticated. If only a log into a system is authenticated, an attacker is able to use banking services after login authentication. Moreover, sophisticated man-in-the-middle attack may hide the misuse from a user.

However, as the phishing case in Finland demonstrates, one-time passwords will not always provide sufficient protection. Especially, when the phishing techniques will develop and sophisticated methods will be utilised, more protection will be needed.

For example, U.S. Federal Financial Institutions Examination Council (2006, p. 3) recommends using at least two authentication methods from the following:

- *"Something the user **knows** (e.g., password, PIN);*
- *Something the user **has** (e.g., ATM card, smart card); and*
- *Something the user **is** (e.g., biometric characteristic, such as a fingerprint)."*

According to Federal Financial Institutions Examination Council (2006, p. 11) one-time passwords are something that the user has. Although one-time passwords improve security, I would argue that they are still something the user knows. The reason is that a user can simply type in passwords and thus mistakenly may give them to a phisher. Moreover, as illustrated earlier and in Appendix 2, there are successful phishing methods against one-time passwords. In table 1 is presented a classification of some known user authentication methods

*Something the user **knows***	*Something the user **has***	*Something the user **is***
PIN code Password/phrase Shared secrets (see Appendix 1) Virtual keypad (see Appendix 1) Changing passwords Disposable passwords	Smartcard Hardware token device (see Appendix 1) SIM card	Personal contact Biometric identification e.g. • Fingerprint scan • Face recognition • Iris/retinal scan • Voice recognition • Handwriting recognition • Geometry of member of the body (e.g. finger or hand)

Table 1: Classification of known user authentication methods

It would be possible to use all the three authentication methods and still keep the authentication process fairly simple. For example, pass phrase authentication (something the user **knows**) can be combined with a hardware token device (something the user **has**) that uses PKI and fingerprint authentication (something the user **is**). Of course, extra hardware costs. However, usage of authentication could be stimulated by financial benefits.

Personalisation

Emigh (2005, pp. 21-23) finds personalisation as one important prevention method against large scale phishing. Banking sites and e-mails received from the banking site should be personalised so that a customer will detect that the service is likely to be correct. A customer could receive messages that are tailored according to his/hers preferences. Accordingly, the service can be tailored. Although personalisation does not prevent from man-in-the-middle or user-end-hijacking attacks, correctly implemented personalisation could make massive fraud attempts difficult to implement.

Fraud web service detection and prevention

Once a phishing site is established it is available in the Internet. Therefore, it is possible to detect the sites based on content, suspicious references and fraud messages. Financial institutions can use and support services that detect and bring down fraudulent services. Furthermore, it is possible to register domain names resembling own financial service. Finally, it is possible to follow registrations and take actions against resembling domain name registrations.

Preparation

It is important to prepare for an attack in advance. Emigh (2005, p. 15-16) proposes how an organisation can prepare for a phishing attack. Such preparation includes spoof-reporting service, monitoring bounced e-mail messages, monitoring customer service enquiries and volumes, monitoring anomalous account activity, monitoring use of service content and honeypot systems.

Of course, there must be a security policy that takes in account phishing and the policy must be known, maintained, watched and followed. Therefore, for example, education that takes in account phishing is needed for the personnel.

Furthermore, design of the services is part the preparation. For example, SSL should always be used, e-mail communication should be consistent and personalised and should not include clickable links while domain names should be consistent and easy to understand. All e-mail communication should be digitally signed so that customers have a possibility to authenticate the sender.

Raising awareness

Banks are in a key position for raising customer awareness. Therefore banks should inform customers about security policies and current threats. Customers must be aware of the possibility of phishing and they should be provided information about preventive actions. Policies should be regularly and constantly informed to customers. Meanwhile, it must be realised that as long as systems allow misuse raising awareness cannot be the definitive solution.

Network logs

Network and system logs should be gathered as precisely as possible. Network logs should be utilised for tracing the sources of malicious attacks associated with phishing. Whenever possible, network logs should be printed or written on a write-only storage so that intruders cannot modify the logs.

Authority perspective

Authorities have legal guidance and can affect incident handling. We will briefly discuss authority perspective.

Reaction

It is important to react to phishing incidents as quickly as possible. As the local example (see Appendix 2) demonstrates, efficient authority co-operation may produce some results. Furthermore, authorities can prepare in advance to quick actions whenever necessary. For example, money transactions can be traced, fraudulent accounts freezed etc.

Legislative guidance

Authorities have also legislative guiding power against phishing. As Leveson (1994) observed, legislation may be needed in order to prevent accidents. Accordingly, authorities have the possibilities to affect on responsibility and to limits how on-line banking is allowed to behave.

World-wide co-operation

Because Internet is a world-wide domain also solutions should be world-wide. It is essential that there exist co-operation for authorities across national borders. Although, admittedly, achieving co-operation is troublesome, there should be serious aims for co-operation. I see co-operation essential especially in the areas of legislation, network forensics and international encryption standards.

DNS-registration

Although it is difficult to control domain name registration, there should be methods to prevent fraudulent registrations from the authority perspective. For example, domain names resembling known financial services could be watched with special care.

Conclutions

We discussed some reasons for the phishing problem and proposed a classification of phishing methods. We found that there are methods to reduce the phishing problem. Key improvements seem to be restrictions to user accounts, proper authentication, personalised services, preparation and legal guidance. Of course, all the factors affecting the problems and all solutions are not included in this paper and it is worthwhile to research and develop the issues further. We will finally propose some recommendations for system designers and decision-makers and propose some valuable research areas.

Secure authentication

Encourage for secure authentication is needed. One potential solution for authentication is correctly implemented smartcard technology, where encryption software and secret keys are physically protected. Usage of smartcard technology for authentication could be stimulated by financial benefits. Indeed, in some countries the infrastructure is ready for national authentication (e.g. Estonia and Finland).

Futhermore, Emigh (2005, p. 35-38) proposes technologies where a trusted path is established between a customer and a service provider. The idea is that user-end-hijacking should not affect the authentication process. Emigh proposes that there could be special feature for using trusted path in operating system level and by activating the feature information can be transmitted securely. However, if there is a vulnerability in the operating system, the trusted path may be compromised. If more security is needed there should be hardware level security for creating the trusted path.

Hardware level security

Current systems were not designed security in mind. By providing support for hardware level security also the phishing problem could be restricted. For example, secure biometric authentication, secure keypad systems or smartcard readers could be part of computer systems. Of course, hardware level security could improve system security as well (Helenius 2003). For example, system integrity and user right control could be implemented more securely.

Proactive filtering

There should be proactive filtering of malicious network traffic in a more proactive way. Malicious network traffic includes, not only malicious code, but also other malicious content (e.g. spam, denial of service, hacking and phishing content).

Network operators are in a key position for recognising and filtering malicious network traffic. Network operators should have the legal right, obligation and technical means for isolating such systems that cause malicious network traffic.

Furthermore, e-mail traffic could be filtered more aggressively in case of phishing. Technically it would be feasible to mark obvious phishing e-mail as spam whenever encountered. This could be achieved by centralised spam control. Furthermore, e-mail clients could be developed so that they are able to filter already received phishing e-mail.

Of course, it must also be realised that filtering cannot be the final solution. Proactive filtering is, however, a necessity for restraining the phishing problem in the current infrastructure. Furthermore, in many legislations there could be more efficient guidance for proactive filtering. For example, legislation could obligate for proactive filtering.

Resources for research

Research creates possibilities to improve information security. For example, resources for the following areas could be directed.

- Development of secure network protocols. For example, packet level encryption and IPv6.
- Research of hardware level security solutions
- Research investigating phishing incidents.
- Research of network forensics and intrusion prevention
- Research of security in electronic commerce in general.
- Research of authentication methods.
- Research in international fraud prevention.
- Research of legislation associated with phishing

References

Anti-Phishing Working Group (2006), Phishing Activity Trends Report, November 2005 http://www.antiphishing.org/, Retrieved 13 January, 2006

Bellovin Steve (2004); Spamming, Phishing, Authentication, and Privacy; Inside Risks 174, CACM 47, 12, December 2004

Bontchev Vesselin (1998). Methodology of Anti-Virus Research. Dissertation. Faculty of Informatics, University of Hamburg, 1988

Casescontact (2006), 10 commandments for more secure online banking, http://casescontact.org/tips/210014, Retrieved 13 January, 2006

Ellison Carl and Schneier Bruce (2000) Risks of PKI: Electronic Commerce, Inside Risks 116, CACM 43, 2, February 2000

Emigh Aaron (2005), Online Identity Theft: Phishing Technology, Chokepoints and Countermeasures, http://www.antiphishing.org/Phishing-dhs-report.pdf, Retrieved 11 January 2006

Ernst & Young and TRUSTe (2005), "How Not to Look Like a Phish - Tips to Help Your Organization Minimize False Positives", Retrieved 16 March, 2006, http://www.truste.org/pdf/How_Not_Look_Like_Phish.pdf

Helenius Marko (2003). Realisation Ideas for Secure System Design. In U.E.Gattiker (Ed.), EICAR Conference Best Paper Proceedings (ISBN 87-987271-2-5) 10 pages. Copenhagen: EICAR

Hoglund Greg and Butler James (2005), "Rootkits : Subverting the Windows Kernel", Addison-Wesley, ISBN: 0321294319

Howard (2004), Browsing the Web and Reading E-mail Safely as an Administrator, November 15, 2004, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncode/html/secure11152004.asp, Retrieved 13 January 2006

Kasslin Kimmo, Sthålberg Kimmo, Larvala Samuli and Tikkanen Antti (2005) Hide'n Seek Revisited – Full Stealth Is Back. In proceedings of the 15th Virus Bulletin International Conference, 5.-7.October 2005, pp.147-154

Lance James (2005), Phishing Exposed, Syngress Publishing, ISBN: 159749030X

Layden John (2005) "UK banks hope to send phishing mules packing", The Register, 24th May 2005, http://www.theregister.co.uk/2005/05/24/phishing_mules_security_checks/, Retrieved 17 March, 2006

Leveson Nancy (1994), High-Pressure Steam Engines and Computer Software, IEEE Software, October 1994, Retrieved 18 March, 2005, http://sunnyday.mit.edu/

Muttik Igor, (2005) "Manipulating the Internet", In proceedings of the 15th Virus Bulletin International Conference, 5.-7.October 2005, pp. 18-25

Onkalo Minna (2005), "Sähköisten maksutapojen hyväksyminen yritykseltä kuluttajalle suunnatussa kaupankäynnissä", Turku School of Economics and Business Administration, Master's thesis

Roberts Paul (2004), “Gartner: Phishing attacks up against U.S. consumers”, Computerworld, 6 May 2004, Retrieved 17 March, 2006 http://www.computerworld.com/securitytopics/security/cybercrime/story/0,10801,92948,00.html

Schneier Bruce (2005), Two-Factor Authentication: Too Little, Too Late, Inside Risks 178, CACM 48, 4, April, 2005

Spamhaus (2006). The Definition of Spam. Retrieved 22 March, 2006, http://www.spamhaus.org/definition.html

Tocheva Katrin and Rautiainen Sami “Scripts - Appetizer or Main Course?” EICAR 2005 ConferenceBest Paper Proceedings, ISBN: 87-987271-7-6, EICAR e.v., pp.

Appendix 1: definitions of some terms

changing password: A password that changes after being used. In distinction to disposable passwords changing passwords will be recycled.

computer virus: Refers to a program code which has a capability to replicate recursively by itself. Computer viruses may include operations, which are typical for Trojan horses and malicious toolkits, but this does not make such viruses Trojan horses or malicious toolkits.

computer worm: Refers to an independent program code which has a capability to replicate recursively by itself. Independent means that a computer worm does not have a host program which the worm has infected or replaced by its own code. Computer worms are a subgroup of computer viruses. Because the definition of a worm is not commonly agreed, I have preferred to use term virus instead.

disposable password: A password that can be used only once. When the password has been used it will not be usable any more. Typically a user has a list of passwords to be used and after all the passwords have been used the user receives a new list.

fraud web site: A fake web site that is used for imitating real e-commerce service. From a fraud web site the criminal's aim is to obtain critical user data. Typically a fraud web page contains images from the real service and it is in the criminals' interest to make the fraud web page resemble as much as possible the real service.

hardware token device: Refers to a device that is needed for authentication. Without device authentication is supposed to fail. Such device may encompass e.g. encryption software, secret keys, keypad and/or biometric identification.

keylogger: A Trojan horse that will record keystrokes. The keystrokes may be sent via Internet connection.

one-time password: The same as *disposable password.*

shared secrets: Shared secrets are some information that both the service provider and the customer knows, but an attacker is not supposed to know.

spam: Unwanted and without users' permission massively distributed e-mail. Often used wording is "unconsolidated bulk e-mail". We must note that in these definitions there are imprecise terms. It is difficult to say what is massive distribution or bulk e-mail. In addition, it is difficult to say what is unwanted. What is spam for one person may not be for some other. In fact, there does not seem to exist precise commonly accepted definitions for spam. Further information about spam definitions can be found at the Spamhaus project's Internet page (2006).

Trojan horse: Refers to a self-standing program code which performs or aims to perform something useful, while at the same time intentionally performs, unknowingly to the user, some kind of destructive function (constructed from Bontchev's (1998, p.14) definition). Self-standing means that, in distinction to viruses, the program code does not have the capability to replicate by itself. The program code may be attached to any part of a system's program code. Trojan horses may include operations which are typical for malicious toolkits but this does not make such Trojan horses malicious toolkits.

two-factor-authentication: authentication which uses some disposable part (e.g. part of the password is always changing). Disposable passwords can be part of two-factor authentication.

virtual keypad: Refers to keypad displayed on the screen. A user will click desired characters from the screen. Virtual keypads can be used to make it more difficult to hijack information submitted to a service. Virtual keypads may be efficient against keyloggers, but not against screen capturing and sophisticated man-in-the-middle attacks.

Appendix 2: The First Phishing Case in Finland

In December 2005 a phishing e-mail targeted to Nordea customers was sent several times to Finnish e-mail addresses. It was estimated that roughly 500 000 messages were sent each time (the population of Finland is 5.2 million people while the exact number of Nordea on-line banking customers in Finland is unknown). Nordea uses a user ID and one-time passwords for log in. Changing passwords are used for confirmation of each critical operation.

The phishing method was typical: A user was first misled to a fake web page and was asked to fill in passwords. At two first times the phishing e-mail was written in English and according to Nordea there were no successful frauds. After that the message was sent in poorly written Finnish. There were about 20 customers who filled (successfully enough) their password lists. From these customers the bank accounts were emptied and the money was transferred to bank accounts in Latvia. In some cases the transfers were cancelled and the money could be recovered. In 7 cases the criminals were able to fraud money. Total amount of the money lost was according to Nordea about 60000 euros. While the bank promised to compensate the losses to the victims', in the future the bank will reconsider compensation, in case a customer gives his/her password lists to outsiders.

While outside from Finland phishing against banking has become a real problem, this was the first successful attack in Finland. The fraud received large publicity in Finland. Therefore banks, authorities and consumers have become more aware of the problem and the phishing problem in general. Although there were successful attacks, efficient authority co-operation was able to prevent some of the fraud attempts. The attack itself was trivial, yet the criminals were able to gain results. If there will appear cleverer phishing attacks consumers are likely to be more vulnerable.

Identifying and Preventing Email-Based Virus Outbreaks

Dirk Beste
IronPort Systems

About Author

Dirk Beste, Dipl.-Inf. (FH), Systems Engineer, IronPort Systems

Dirk Beste, having started mid 2004, has been one of the first employees of the Central & Eastern European hub of IronPort Systems. Holding a position as Systems Engineer, Dirk is mainly responsible for a smooth implementation of all IronPort products at the numerous customers. Dirk brings along more than 6 years experience in implementing Secure Mail Gateways at companies of all sizes.

Before starting at IronPort, Dirk held a similar position at the Content Security specialist Clearswift, after successfully graduating from Fachhochschule Darmstadt with a computer science degree. Dirk wrote and completed his diploma thesis in the practical environment of DaimlerChrysler.

Contact Details: IronPort Systems, Bretonischer Ring 13, 85630 Grasbrunn, Germany,
phone +49 89 452227 18, fax +49 89 452227 100,
e-mail dbeste@ironport.com, web http://www.ironport.com

Keywords

Prevention, Virus, Outbreak, Virus Outbreak, SenderBase, VOF, IronPort, Reputation, Filtering, E-mail, Security, Appliance, Global Traffic Data, SMTP, MTA, MGA, Bot networks, Spam, Zombie, SBRS, SenderBase Reputation Score, WBRS, Web Based Reputation Score

Identifying and Preventing Email-Based Virus Outbreaks

Abstract

Today, viruses spread in a matter of hours, often completely disrupting email service for corporations and creating an all-hands emergency situation for IT staffs around the world. In these times of fast proliferating viruses, traditional anti-virus defenses that rely solely on signature-based filters are no longer adequate. While these solutions are very accurate at detecting known viruses, they depend on receiving regular signature updates to protect against new threats. This makes them vulnerable to reaction times that can vary from hours to days. And during that "vulnerability window", a modern virus can propagate globally, bringing email infrastructure to a halt.

Introduction

As virus writers create increasingly sophisticated malicious code and find ever more effective methods to propagate, enterprises find themselves scrambling to keep their networks, servers, and end-user computers safe from new threats.

Traditional anti-virus applications work by searching the contents of files and looking for a recognized pattern of data (a "signature") that is the virus program itself. However, virus writers have come up with various methods to escape detection by changing their programs, making it harder for virus scanners to recognize them as viruses. Today's viruses are either polymorphic or metamorphic and can actually change themselves as they propagate. The increasing sophistication of malicious code is therefore making pattern recognition technologies less and less effective.

This decline in effective virus defense is taking a toll on businesses. The often long delay between the time when a virus attack is launched and when a signature is available can result in hundreds of thousands of infected messages being delivered to enterprise networks and communities of ISP users. Even when the end effects of the virus are minimal, such widespread infection results in major costs. Productivity is lost as employees try to understand what's wrong with their computers and seek help. Clearing computers of viral infections requires both manpower and other resources to aid in the clean-up. That translates into tens or even hundreds of thousands of dollars in desktop clean-up costs for each virus outbreak at each corporation. If there has been actual data destruction, the costs can be immense, possibly immeasurable.

Another factor contributing to the increasing seriousness of computer virus attacks is the motivation of today's virus writers. Although teenage hackers looking for ego gratification still exist, anti-virus research groups are seeing an increasing number of viruses designed to bring in money, sometimes as part of criminal activities. Political and social extremists have also turned to computer viruses as one way to accomplish their goals.

Spammers, the people who send out millions of unsolicited commercial email messages a day, realized at some point that virus writers could help them overcome spam-blocking measures on corporate and ISP networks. Some viruses are now designed to turn network-connected PCs into robots for sending spam. Most of these robot PCs (or "zombies") are in homes, but many also get created in enterprise environments. Such hijacked computers now make up the BotNets responsible for most of the world's spam email messages. Many of those are fraudulent or "phishing" email messages that trick unsophisticated recipients into revealing personal information, such as passwords for financial accounts. See Figure 1.

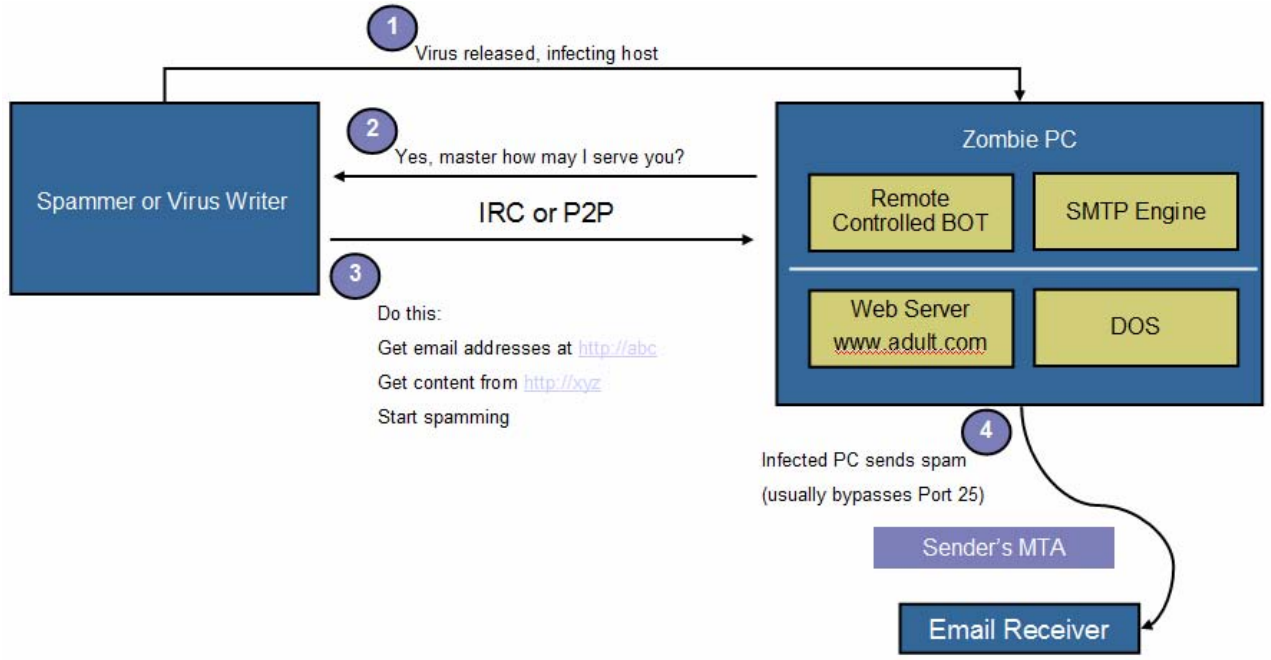


Figure 1 A typical virus/spam attack

There are also organizations in the world who wish to disrupt Western economies. They realize that a virus that could potentially wipe out the hard drive of every infected PC could cause an economic impact of billions of dollars in a matter of hours.

An estimated 900 million virally infected messages a day are currently coursing through the world's email networks, and the problem is increasing dramatically. As a result, many companies and vendors are looking beyond today's signature-based anti-virus solutions and exploring preventive systems that can stop virus outbreaks before they happen.

Discussion

Emerging Anti-Virus Strategies

With the decline in effectiveness of traditional pattern recognition technologies, developers of anti-virus solutions are combining several new approaches to address the problem of increasingly sophisticated computer viruses. These new technologies include heuristic filters, behavioral analysis, and traffic data analysis.

- Heuristic filters are based on artificial intelligence techniques, so they become more accurate as they learn which messages contain viruses and which do not. Unlike pattern scanners, heuristic filters can detect a virus that hasn't been identified yet, so they can stop infections before a signature is released. However, their catch rate is significantly below 100 percent, and they are subject to false positives, which can prevent organizations from receiving vital legitimate business email that has been incorrectly identified as having a virus.
- Behavioral analysis systems actually load and execute a program attached to an email message (or downloaded from a web link embedded in a message) and analyze its behavior as if it were running on an end user's computer. The system can either emulate execution of the program or run it on a separate virtual computer (usually called a "sandbox"). The

behavioral approach can be effective, but it is very resource intensive and not easily scaled to enterprise levels.

- Traffic analysis solutions are based on the fact that virus outbreaks come in waves of email messages, so there are patterns of email traffic anomalies associated with an outbreak. Experienced computer security personnel can detect these anomalous traffic patterns and relay the information to security devices. Because this approach requires a large, global dataset in order to identify patterns as they emerge, only security companies that are monitoring a significant number of large networks for enterprise and ISP email traffic are capable of using this technique.

While all three of these approaches hold some promise of greater virus control, traffic pattern analysis is widely regarded as the most promising technique. It works regardless of message or program content, and eliminates dependence on the ability of a certain computer system to recognize a virus program. This is important because virus writers use morphing algorithms to confuse pattern or signature-based systems by changing how they look and behave, and even where they are housed. Ironically, the more effective a virus is at avoiding detection by traditional signature-based filters, the faster it will propagate globally, and the more quickly a recognizable viral traffic pattern will emerge.

A Comprehensive Approach

Traffic data analysis is emerging as the best technology for detecting viruses quickly and accurately. However, it must be supplemented by the right infrastructure and supporting technologies if it is to offer truly effective virus defense for organizations around the world.

An accurate and efficient predictive virus solution should include:

• mechanism for gathering global data on email traffic.

• threat operations center with highly trained personnel who can detect emerging threats from analyses of traffic patterns.

• The ability to quarantine suspicious email messages based on dynamically changing rules.

Global Traffic Data

The key ingredient in creating an effective traffic-based virus detection system is a world view of email traffic patterns. The best solutions have a very large database of email traffic. These databases need to have messages from ISPs, enterprises, small and mid-sized businesses, education, health-care, and government, just to name a few.

Morphing viruses such as Sober, SoBig, Netsky, and Bagle have propagated rapidly because there were no preventive signatures in place for a long period of time. They caused major disruptions to corporate and ISP networks, and in the process, created huge anomalous patterns in worldwide email traffic. It is impossible for normal human email messaging to create traffic patterns like those in which a single virus program spreads around the globe in two hours or less. So gathering real-time data from around the world is an essential foundation to detecting new attacks.

Systems, such as SenderBase[1], track a variety of network parameters about any given IP address sending mail on the Internet. These parameters (Figure 2) include the global volume of mail sent by any given IP address, how long that IP has been sending mail, country of origin, open proxy or open relay detection, appearance on any black or whitelists, proper DNS configuration, ability of the sender to receive mail in return, etc.

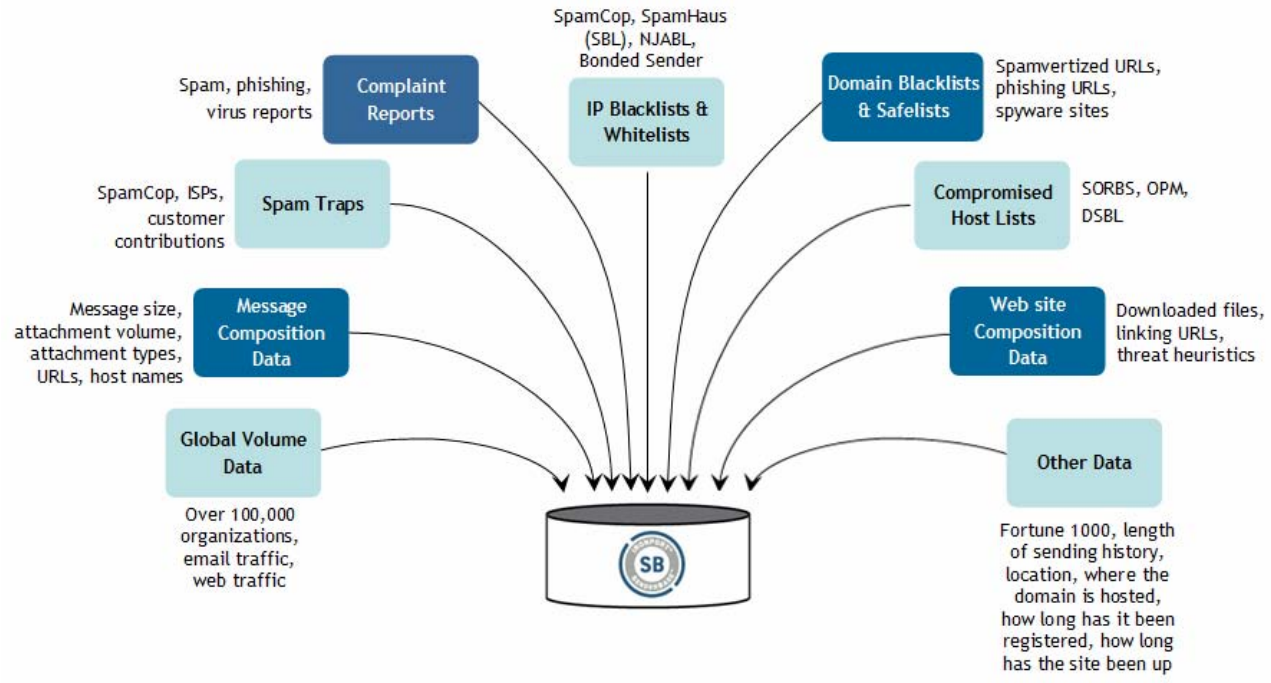


Figure 2 A Global Data Monitoring Framework

These systems usually track a high volume of SMTP data all around the globe and a lot of different parameters about a given sender. By accessing a broad set of data across a very large sample size, these systems are able to make extremely accurate assessments of a sender's behavior and reputation.

Normally the indexes contain data about the IP, Domain and Network Owner level. This is a simple but incredibly powerful way to combat spam. Spammers will often have large networks with multiple domains and IP addresses. If a spam message is received from a single IP address, blocking that IP is a bit like squashing a single ant found in your kitchen. There are literally hundreds or thousands of more IP addresses and domains, akin to thousands of ants in the nest, waiting to attack your network. Also these systems provide a simple tool to root out the entire nest – block thousands of IP addresses across multiple domains all rolling up to the same organization with a single click.

Global Traffic Data allows the administrator to rapidly "zoom out" and examine other mail servers on the same netblock as the server in question. Practical limitations force spammers to operate out of a limited number of co-location facilities. As a result, where you find one spammer, you will often find others on the same netblock. A quick "zoom out" to examine the /24 level of any IP address will often reveal a pattern of servers with high volumes and high complaint rates. Conversely, if you are tempted to block an entire netblock, a quick examination of the entire

[1] http://www.senderbase.org

netblock in a monitoring system can help identify any innocent mailers that may inadvertently be affected, so that you can narrow your block or whitelist those senders specifically.

In addition to volume and network data, some systems aggregate multiple different black lists and provide a consolidated view of whether or not the IP address in question appears on any of them. SenderBase, in partnership with SpamCop[2] and other sites, also contains data on end user complaints and samples of messages sent from the IP address in question. While much of this data is available in a variety of places on the Internet, SenderBase provides a unified view that is indexed to other indexed data.

SenderBase has algorithms that analyze these objective, network level parameters and distill a "reputation score" of -10 to +10. This score is then made available to the appliance in real time as a message is received from any sender. A variety of email policies can be tied to a sender's reputation, such as flow control parameters, attachment size or type restrictions (Figure 3).

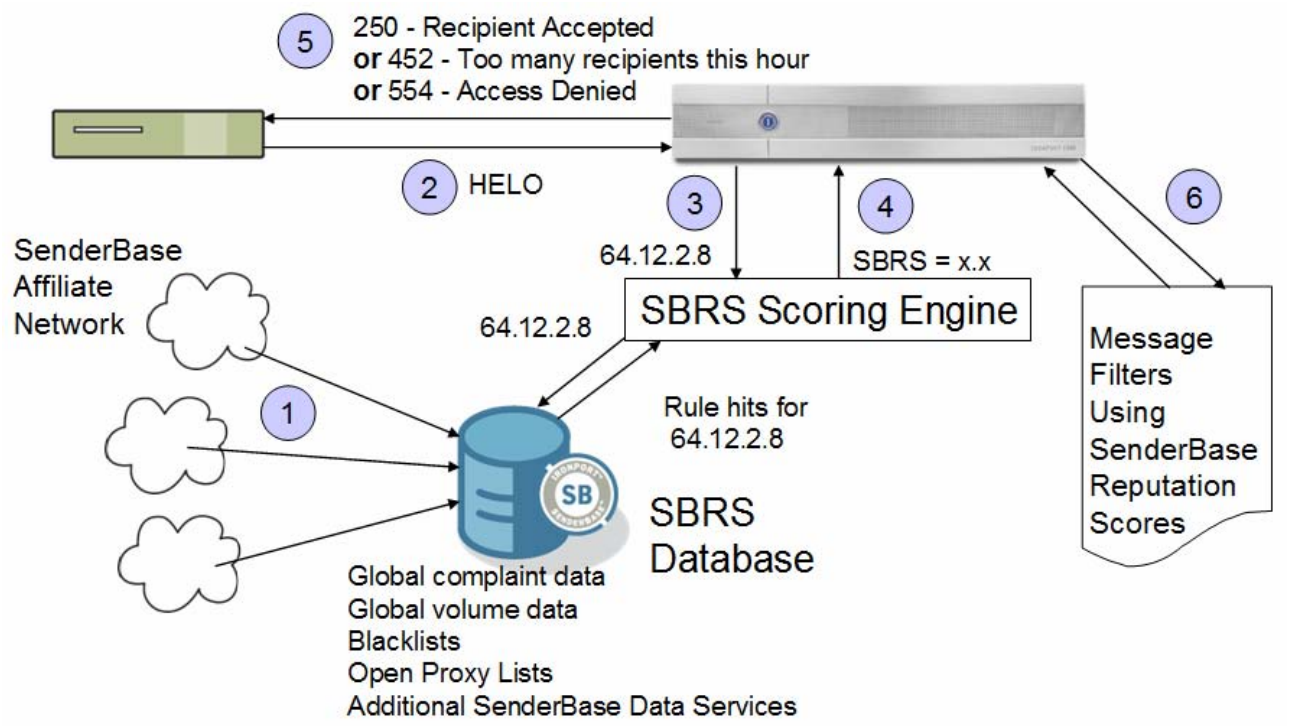


Figure 3 Resolving the Reputation Score

Threat Operations Center

A predictive system, by definition, is responding to unknown threats. Global data and sophisticated algorithms are very powerful tools to combat these threats, but there is no substitute for human oversight in helping to identify new anomalies and new outbreaks. The most sophisticated preventive solution will include a fully staffed 24x7 threat operations center that has multi-lingual analysts and statisticians reviewing dynamic email traffic data. The Threat Operations Center should constantly be looking for traffic peaks such as an unusual amount of ".pif" files or other threats (Figure 4).

[2] http://www.spamcop.net/

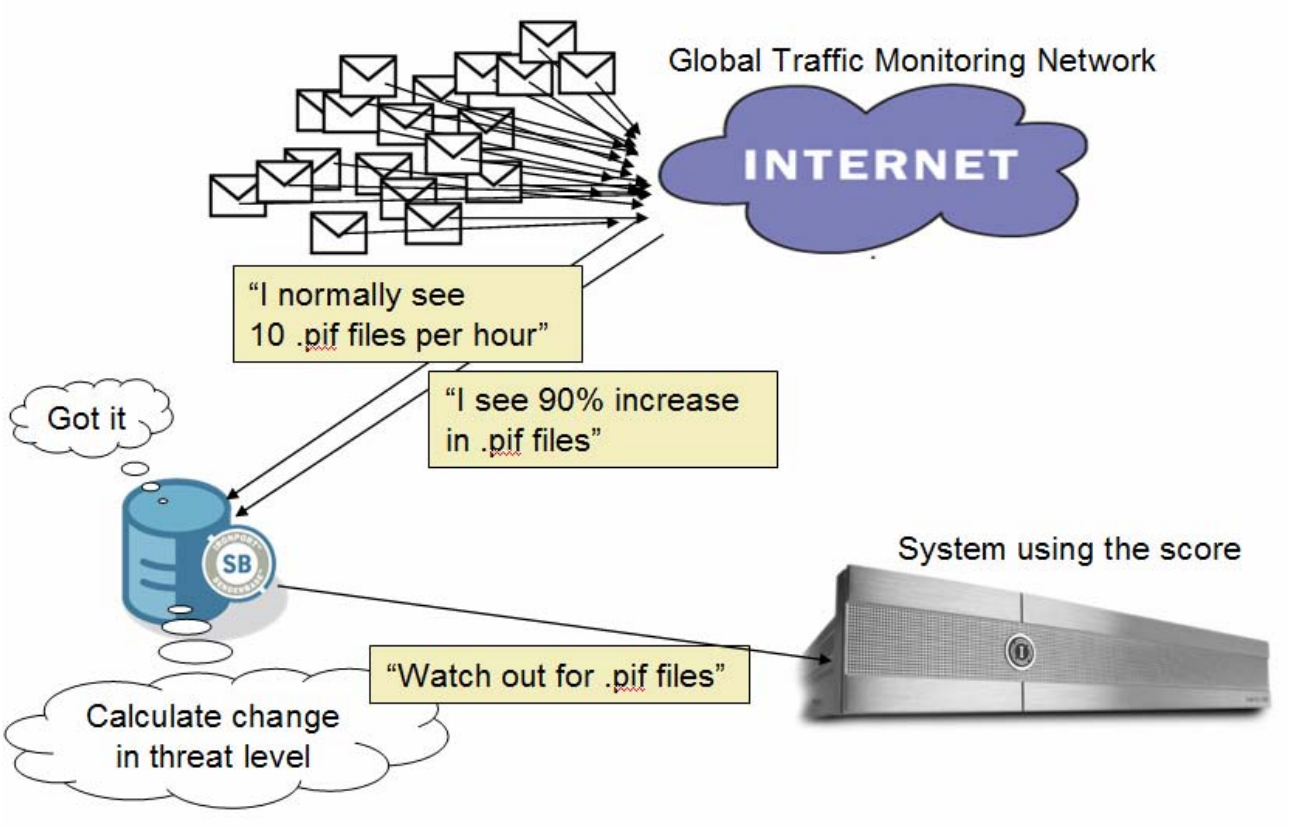


Figure 4 Looking for Traffic Changes on a Global Basis

Dynamic Quarantining

The key concept behind predictive virus systems is that they can take action earlier than traditional systems, but with lower confidence. Thus, a sophisticated quarantine system is an essential ingredient to mitigate false positives. The quarantine software should have tools to allow administrators to easily address exceptions, release certain messages, or "opt-out" certain users.

More advanced systems use a new and very promising technology called dynamic quarantining. This approach offers continuous, automatic rescanning of all messages in the quarantine area. It makes possible the powerful combination of traffic-based anti-virus systems (which are highly accurate but take some time to detect patterns) and heuristic filters (which can react to new viruses immediately but are more subject to false positives).

Dynamic quarantining uses the coarse rules of a heuristics system to stop an outbreak as soon as it occurs, before enough anomalous traffic has appeared to develop a traffic-based rule. Because it quarantines rather than deletes messages, it mitigates the false-positive issues associated with heuristic methods.

At time zero, many messages may get quarantined, adding a slight bit of latency to the mail flow. However, within minutes, new data will emerge that narrows the quarantine and returns known good messages back into the mail flow. Because email is an asynchronous medium, most users will be unaware of the small added latency; it is a small price to pay for extremely robust virus protection.

As soon as additional data on the outbreak becomes available, new outbreak rules are issued in real time. The quarantined messages are immediately rescanned, and the system releases all messages

that do not match the newer, more fine-grained outbreak rule. Additional anomalies and observations will spawn updated rules, which will further narrow the quarantine to only infected messages. An example of a quarantine sequence is shown in Table 1.

Time	Rule	Action
T=0 mins.	Quarantine all attachments whose file type does not match the extension names. (F.e., a file that says it's a .doc but is really a .zip.)	Implement a heuristic rule built from profiles of known outbreaks.
T=2 mins.	Quarantine all .zip attachments that contain an exe file.	Release from quarantine any files with mismatched file type and file extension, except for .zip(exe) files.
T=5 mins.	Quarantine .zip(exe) files that are greater than 50 KB.	Release from quarantine any .zip(exe) files that are less than 50 KB in size.
T=15 mins.	Quarantine .zip(exe) files between 50KB and 55KB that have "price" in the filename string.	Releases from quarantine any .zip(exe) files that are smaller thant 50KB or larger than 55KB, or which do not have "price" in the filename.
T=6 hrs.	Scan against new anti-virus signatures.	Scans all remaining messages against the latest signature file from AV vendor.

Table 1 Sample Rule Sequence for a Dynamic Quarantine[3]

Each rule includes a threat level score. The score ranges from 0 (no threat) to 5 (extreme threat). Threat levels are set based primarily on the certainty that there is an outbreak happening with a particular set of characteristics, and the distribution size of the outbreak, and secondarily on the damage potential of the virus. Table 2 shows a real-life rule with a more detailed ruleset.

Date/ Time	**Rule_ID**	**Threat Level**	**Rule Description**	**Decision Factor**
02/09/06 20:09:27 GMT	OUTBREAK _0000524_06	4	file/container name keywords (^(Generated_bill|Order_details|Service_receipt|phishing_scr eenshot)\.exe$), file extensions (exe), and file/container sizes (5000-250000)	We are seeing unusual volume for this file extension. We are raising the Threat Level for this extension to 4. We will continue to monitor the situation

Table 2 Real-life data from http://www.ironport.com/outbreak_alerts/

[3] http://www.ironport.com/pdf/ironport_preventing_virus_WP.pdf

Conclusion
The Essence of Time

Today's aggressive, intelligent viruses, especially when they're propagated via sophisticated spamming technique, can infect hundreds of thousands of computers in a very short amount of time. Even the fastest virus sleuths take the better part of a day to get a new virus diagnosed and create a signature for it. Then it takes more time to distribute the signature file. Even aggressive signature update schemes leave users and enterprise networks vulnerable to virus attacks for anywhere from twelve hours to three days, more than enough time for a virus to do serious damage.

According to AV-Test[4], a German virus research group at the Otto von Guericke University in Magdeburg, the response times of anti-virus vendors to the emergence of a new virus vary dramatically. The group studies vendor performance by measuring the time from when a new virus is first spotted by a British consulting group, to when each vendor makes a signature file available. AV-Test checks anti-virus databases every five minutes for the presence of a new profile. The results[5] shown in the Table 3 were based on four virus outbreaks: Dumaru.Y, MyDoom.A, Bagle.A and Bagle.B.

Average Response Time (Hours:Mins)	**Vendor**	**Average Response Time (Hours:Mins)**	**Vendor**
06:51	Kaspersky	12:22	Sophos
08:21	Bitdefender	12:31	Dr. Web
08:45	Virusbuster	13:06	Trend Micro
09:08	F-Secure	13:10	Norman
09:16	F-Prot	13:59	Command
09:16	RAV	14:04	Panda
09:24	AntiVir	17:16	Esafe
10:31	Quickheal	24:12	A2
10:52	InoculateIT-CA	26:11	McAfee
11:30	Ikarus	27:10	Symantec
12:00	AVG	29:45	InoculateIT-VET
12:17	Avast		

Table 3 Average Response Time from different AV vendors

[4] http://www.av-test.org/

[5] http://www.ironport.com/pdf/ironport_preventing_virus_WP.pdf

As Table 3 shows, the results for those four viruses vary from nearly seven hours to more than a day. In the virus outbreaks monitored by IronPort Systems in Table 4 , as much as three days have passed for some vendors to find the correct profile. The data does not reflect the fact that heuristic techniques used by some of the vendors allow viruses to be detected and blocked before signatures are developed and published. Nonetheless, even in the quickest case, the potential for a serious amount of destruction to occur while waiting for virus signature files is considerable.

Virus Name	**Date**	**Virus Threat Level Raised**	**First AV Signature Available**	**Outbreak Filter Lead Time**
Looksky.G	6 Jan 06	14:32 GMT	8 Jan 06 @ 2:12 GMT	35:40 hours
Feebs.M	11 Jan 06	9:42 GMT	12 Jan 06 @ 3:30 GMT	17:48 hours
Feebs.E	13 Jan 06	17:23 GMT	13 Jan 06 @ 22:26 GMT	5:03 hours
Nyxem-D	16 Jan 06	14:36 GMT	61 Jan 06 @ 15:22 GMT	1:27 hours
Stinx-T	31 Jan 06	14:39 GMT	31 Jan 06 @ 18:06 GMT	3:27 hours

References

IronPort Systems Whitepaper, A Multi-layered Approach to Preventing Viruses, http://www.ironport.com/pdf/ironport_preventing_virus_WP.pdf

IronPort Systems Whitepaper, Reputation-based Mail Flow Control, http://www.ironport.com/pdf/ironport_wp_reputation_based_control.pdf

Localization issues of threats and protection. What's the use of protecting me *here* by products developed somewhere *there*?

Michael Kondrashin

Russian Trend Micro Competence Centre

About Author

Michael Kondrashin is head of Trend Micro Competence Centre in Russia.
Contact Details: R&D CALS Centre Applied Logistics, 5-th Donskoy, 21-B, 119991, Moscow, Russia, phone +7 (495) 955-51-85, fax+7 (495) 959-91-03, e-mail mkondrashin@apl.ru

Keywords

Linguistic Vulnerability, Virus, Antivirus, Localization, Spam, Phishing

Localization issues of threats and protection. What's the use of protecting me *here* by products developed somewhere *there*?

Abstract

Under certain circumstances, unlocalized and uninternationalized versions of antivirus products present threats just by not supporting the local language. In the present paper we give examples of threats, caused by such "linguistic vulnerabilities" of antivirus products and other software developed abroad. Our aim is to point out to developers worldwide that all modern Internet-era threats are global and protection tools should be developed using global thinking.

Introduction

Once upon a time computer viruses were spreading locally. During the MS DOS era all viruses migrated on floppy disks and this set natural limits on their habitat. Non local antivirus research laboratories had no chance to receive samples from Far Away for analysis. At those times foreign antivirus programs were almost useless.

During the eighties, thanks to the Internet and Microsoft we all started to suffer from the same viruses, and this gave opportunity for antivirus companies to extend their business onto global market. The basic idea behind it was simple: the same threat – the same protection. Is this assumption true? It seems that things are changing**.** In this paper we try to present some disadvantages of the global approach as seen from our perspective here in Russia.

Local viruses – local vendors

As I have said in the introduction, computer viruses were spreading locally at the "Stone Age" of the Internet. The best cure from them was some local antivirus product. Later on, Internet connected all of us into one global network with no borders and viruses started spreading globally. Nowadays, it looks as if things were changing back. Global viruses written by "script kiddies" still remain, but a lot of new malicious code is written by professionals that intend to gain some profit from their creation. For example one can write a Trojan that is stealing Internet access account data. Obviously, this Trojan will be sent only to local e-mail addresses. What happens next? Some of the potential victims upon receiving this "ClickMeAndYouWillSeeMissUniverseNaked.exe" will send it to their antivirus vendor for analysis. This gives the most popular local vendor a chance to create an update for their customers even before this attack has finished. Meanwhile, other "far away" vendors can relax and have a quiet holiday.

Antispam

First of all let us have a look at the local spam problem. Most of the antispam developers are from English speaking countries. When their products came to the Russian market their local spam filtering abilities appeared to be very poor.

In Russia, we have localization problems somehow similar to those in the countries that use double byte character sets. In our case we have more than one character coding due to historical reasons. The most popular character codings are: CP-1251 used in Windows applications; CP-866 which was mostly used in MS DOS and till now is the default character set for MS DOS command line in Windows; KOI8-R which is a USSR development used in most unices (Операционная система UNIX во множественном числе), sometimes in e-mails and on some Web sites. The described variety of encodings makes non Russian content scanning software useless or not easy to adapt. This problem mainly refers to the mentioned antispam software.

For accurate spam filtering, an antispam filter combines a variety of content analysis methods. Obviously, this analysis is tightly language dependent. Can we block our local spam just by using a locally developed antispam filter? In Russia about 70% of the spam received is local, but the rest is mostly English and sometimes even Chinese. This means that even a 100% effective antispam filter for local language is useless: 30% will pass this filter freely. As the result, to develop an antispam product, one needs to implement linguistic analysis not only for the local language, but for English too, as well as for Chinese and all other languages!

Antivirus Notifications

The most important thing that we see from antivirus is notification upon virus infection. Clear understanding of these messages is important for good protection. A foreign language message can be confusing or even pose a threat! Here is a real life example. Let us consider the following message from antivirus: “FOO virus found. Clean failed. Deleted”. Such message can be misinterpreted in the following way: Remove this particular antivirus and then find and install the one that does not fail to clean viruses! All we understand is that in the case mentioned above message should be interpreted as “Virus is uncleanable (it is, for example, a Trojan). The file is not *infected* – it is wholly malicious and should be deleted”, but first of all we know and understand English and it may be not so obvious for people who do not know English quite well.

As you know, some of end users overestimate threat from viruses. This creates disadvantages of antivirus products that just show unlocalized messages on virus detection. Some company administrators even turn this virus notification off. Why? The problem is that when a regular employee sees an unreadable message with the well known word “virus” he stops working for the rest of the day trying to hide the fact that his computer is infected. Unfortunately, disabling warnings is not a good idea, because in this case, the computer’s behaviour will be very suspicious – some (infected) files will disappear upon saving them from e-mail messages, or copying from network shares for example. This will be treated as some kind of bug from the user’s point of view and he may try to get this file by other means. For example, from a colleague’s computer. This is much worse than an unreadable warning message, but it would be much better if all messages seen by the end user were fully customizable. In this case the administrator could provide some explanatory text suitable for his particular organization.

Localization Bugs

We have seen instances of various bugs in installation of antiviruses on localized versions of Windows or using in non English environment. For example infected files with Russian name on local disks or attached to e-mails. Most of these problems are already in the past, but every time you face a new product or a new version of a well known product you should ask yourself whether it was tested in your environment.

Less threatening but more frequent are notifications of log file entries with unreadable names of the files which were detected to be infected. The same applies to localized used names for messaging protection products i.e. antiviruses for Microsoft Exchange. In this case the administrator can see that someone is sending viruses, but can not tell who it is and can not implement appropriate countermeasures.

Localized Patches

Those days, when patches and hotfixes were installed just to fix bugs are gone. Nowadays, most important hotfixes are used to patch vulnerabilities. The latter can be an infection vector for the nearest future threat. But patches should be also localized if the product they are created for has localized versions! Sometimes English patches come first and localized versions come later, leaving users of the product’s localized version vulnerable for a long time.

Phishing

Phishing is another example of a typical domestic threat. If you are a phisher and you want to convince your potential victim to give you some sensitive data, you should know your victum very well. Most popular type of phishing attack is aimed at customers of CityBank. The spammed e-mail looks like some kind of automated CityBank notification and if you click a link provided in this message you are directed to a hacker owned site that looks and feels like the original CityBack Web site, that supposedly is familiar to the victim. The point is that these e-mails are phishing for English speaking customers of CityBank, but for rest of the world they are just spam. This seems like good news for foreign language speakers, but what about local phishing attacks? If a foreign company claims that their gateway protection software gives me protection from phishing, do they mean that they have employees speaking my language who are in charge of preventing my domestic phishing? As I have written before, phishing intended to English speaking victims is not a threat for me at all!

Education

As Trend Micro's Global Director of Education David Perry believes, four things should be done to improve today situation: Enhance operation systems and protection software, create appropriate laws and educate people. The last point is really important but not fulfilled by anyone so far. Education is much simpler than sophisticated software solutions for some kinds of threats. For example, as protection from phishing. It is quite a headache to develop an antiphishing protection, but it is much easier to show a contextual warning to the user when he tries to fill sensitive data into a suspicious Web form. This warning should describe what is phishing and what should be checked before typing in any personal data.
Here is another example of using such “education channel”. As you know, last year Sony have released audio disks with dubious antipiracy protection using rootkit methonds. This program installed itself as a system driver, which could not be easily uninstalled. During the uninstallation Windows could even crash. What if some antivirus company would want to add the uninstallation feature of this Sony Music protection to its pattern files? The end user would see (i) virus found message, probably (ii) blue screen of death or reboot and then will notice that (iii) the disk did not play anymore. Who would be guilty? The antivirus! Antivirus research labs can only show virus name and additionally give some description in the online virus encyclopedia and this is it. What if antivirus had “education channel”? The virus research lab could describe the situation to the user, recommend him to remove this software and describe the consequences. What is the reason that nobody has implemented this kind of functionality in their product? Localization! All this description is useless in foreign language and if some company would want to implement it, it will have to translate hints and warnings it all other languages.

Encyclopedia

Many people are still waiting for some super technology that will protect them from most of the viruses without need for frequent update of pattern file. I do not intend to speculate about this utopian hope here, but I would like to point our one very important role of the good old virus pattern approach. Unlike proactive protection that gives us information on how some malicious code tried to attack us, with pattern based detection we have knowledge what this virus was. If we refer to the encyclopedia of our antivirus vendor, we can tell what was it’s payload, i.e. what could be damage of our system, what other infection vectors this particular malicious code uses to spread, and consequently we can check potentially vulnerable computers on our network. This is very nice, but what if I do not know well the native language of my antivirus developers, which they are using to populate their virus encyclopedia? Bad for me.

Conclusions

We all live on one planet. We are all using the same computers. All of our computers are connected to the same network.
We suffer from the same threats. But we do not all speak the same language. This makes for the difference in threats and in protection they require.

Although various protection technologies were developed during last twenty years most of them should not just be localized, which is important in itself, by also be generalized for a variety of languages. My checklist for any current and future anti-malware technology is as follows:

- If the product processes some named object (files, mail boxes), does it support localized names? Will it display them correctly in GUI, logs, notifications?
- If one is collecting malicious code for analysis, is it done globally?
- If the software has some on-line encyclopedia, is it translated to all languages popular in our target languages?
- If the product performs content checking, will it work with all languages?
- If theproduct has localized versions do we provide patches for all versions at once?
- Are all the messages shown to the end user fully customized? Does the product suit all langauges?

If the answer to at least one of these questions is “no”, then under some circumstances it can be a threat – a Linguistic Vulnerability.

Malicious Packages Based on Legitimate Software.

Taras Malivanchuk
Computer Associates

About Author:

Malivanchik Taras is a Senior Software Engineer with CA in Israel.
Contact Details:
CA Building , Arie Shenkar St. ,p.b. 2207, Herzelia Pituach,46120, Israel ;
taras@iris.co.il.

Keywords

IRCFlood, ServU, Package, Trojan, dropper, removal, registry ,selfextract, bot.

Abstract

This paper considers Trojan packages that are based on legitimate software, such as ServU and mIRC32. These packages usually arrive as self-extracting archives that install a number of executable, batch and INI files . This paper examines methods of installation; worm, Trojan, and DoS functionality; methods of detection and removal, and associated challenges. The paper also contains general classification and detailed analysis of a number of examples from the wild.

Introduction

This paper examines a class of malware referred to here as 'malicious packages'. These packages often arrive as self-extracting archives or may be downloaded as an archive from a malicious site and then installed on a victim's machine.

An affected machine may contain a number of clean and malicious executables, DLLs, associated data files and registry entries. Unlike the case of a single Trojan, it is more difficult to remove all of the package's components manually. It can also be difficult to produce definitions for antivirus, as the relationship between the files and their malicious nature is not clear, and there is a risk of inadvertently damaging clean components or of missing malicious content. There may also be performance issues involved in producing appropriate detections for self-extracting archives. Finally, the same malicious package (or variant) often has a relatively limited distribution in the wild, so the specific detection of particular droppers is less useful or effective (although there have been cases of massive infection by the same malicious package).

Discussion

Examples and analysis

Example 1.

The first example we will consider in detail is a typical MIRC package.

The dropper is a GSfX self-extracting archive that contains files packed in a CAB archive. When extracted, it drops a number of files to a subdirectory that it creates in the System directory. We will consider these components one by one.

Sys.exe is a modified version of MIRC32 v5.91. The names of data files have been changed, as has the icon and the menu items in the body of the executable. 'mirc.ini' has been replaced with '*darkhell*'. This is a typical method of modifying *MIRC32.exe*. The file is not compressed or encrypted. From an AV standpoint this may cause some performance problems: to determine if the file is malicious, we should reach resources near the end of the file. However, this is not a big problem, as there are few occurrences of *MIRC32.exe* on user's machines. Other modified versions of MIRC32 are sometimes compressed or encrypted. This is better from an antivirus point of view, because a packed mIRC32 is more likely to be a Trojan component. Perhaps with very few exceptions, many packed or encrypted executables are in fact malware.

We can examine the names of INI files in the data of *mirc32.exe* to determine if it is potentially a Trojan component. First, we check the CRC of the code to determine which version of MIRC32 is being used. Then, if it is a known version, we go to the RVA of the string and compare it with the original one. Although, keep in mind that this method is not quite appropriate for antivirus research: we could incorrectly determine the RVA, or a user may have patched their own MIRC32. To warn users regarding 'suspicious' files is not good practice: we should aim to either define and flag files as clean or as infected (.i.e. malicious).

The following registry key is created to run MIRC32 at each Windows start:

HKLM\Software\Microsoft\Windows\CurrentVersion\Run\daya

The file also runs immediately after the dropper installs it. There are several different combinations of registry keys that may be used to immediately execute the file and ensure that it runs at Windows start. The worst case scenario for antivirus would be a non-modified MIRC32 that is placed in a neutral location (such as in a subdirectory created under C: or under Program Files) and not set in the registry. In these cases, MIRC32 cannot be reliably detected by antivirus as a Trojan component.

With the registry key and MIRC32 being located in a subdirectory of the System directory, we can reasonably detect it as malware: a user would not normally put it in such a place. Other suspicious behaviours include MIRC32 being located in the Recycled or Recycler directory, or in several other combinations of directories, and the file being packed.

This is a method that we use to remove clean components of a malicious package. First we detect by CRC that it is an application typically used in a malicious package, then we check for its location and associated registry keys, and whether the file is currently running.

Here are some utilities that are included in our first example malicious package:

dex.exe – PsExec utility used to execute a program remotely; used to infect a remote system.

bd.exe - hidewnd utility used to hide windows, used to hide any Trojan related window.

kern.exe – process viewer utility

fgr.bat - is an AddShare type batch Trojan. It attempts to create shared drives using a list of hardcoded user names and passwords at a given IP address. If it is successful, it attempts to run the dropper using PsExec to infect the remote machine. This batch file is a good candidate for generic detection.

man.bat - noshare type Trojan. It closes shares opened by *fgr.bat.* Although not obviously malicious, it is typical for MIRC packages and may also be detected generically.

darkhell - *MIRC.INI* used by modified MIRC32.

a.q, *finger.ins*, *s.c* – include files for *darkhell.*

darkhell does not have obvious Trojan features that help us to detect it, similar to other *MIRC.INI* files used with these packages. However, the include file contains event handlers and can be detected as malware, and if it is deleted, the Trojan ceases functioning.

Consider now *finger.ins* that contains MIRC event handlers. On start it runs the *hidewnd* utility to hide the MIRC32 window. This is a standard procedure for Trojan packages and may be detected in the file. On BNC Client port it provides a BNC (Bounce, IRC Proxy) interface. The script accepts Trojan related commands: IP scanning using the fore mentioned *fgr.bat*, BNC, file downloading and uploading.

Fun.exe – the dropper that initially arrives at the victim's machine and is used to propagate further to other machines where the Trojan succeeds in creating shared drives. It appears as follows: It starts with unpacking code, followed by the *GGSfxExecutePack.DLL* that is responsible for executing packed files, then the unpacking execution script, and finally the CAB archive containing compressed files.

The execution script defines the behavior of the self-extracting archive, i.e. where the files will be unpacked, which file will be executed, etc. In addition, we can see the names and sizes of files inside the CAB. Analysis of the self-extracting execution script and files helps to improve performance by helping us to decide whether we should decompress a self-extracting archive (as long as it appears to be suspicious).

Example 2.

This package arrives as a self-extracting RAR archive. It installs files into a subdirectory under the Windows directory: *SYSTEM32/Drivers/etc*
(regardless of the OS: there is no such directory in Windows 9X).

The batch file *start.bat* is then executed. It runs the following components:

- *Firedaemon.exe* - used to run programs as Windows services
- *winserv.exe* - IrOffer
- *svchost1.exe* that is ServU.

IrOffer is an IRC fileserver that allows for downloading and uploading files using the DCC protocol. ServU is an FTP server. They are both legitimate programs and are detected by antivirus as described above for *MIRC32.exe*, that is, by analyzing their locations and the fact that they run as services. Both programs are supplied with INI files that contain the credentials of particular hackers that purportedly use them.

IrOffer joins a channel on a certain IRC server as dictated by the INI file, so the hacker knows that the Trojan is online. The Trojan then runs Radmin: a legitimate remote control program. The package also contains a batch file and an executable used to kill antivirus and other security program processes and files, a utility used to get drive information, and another utility used to get IP addresses.

Example 3.

The dropper in this case is also a self-extracting RAR archive. When executed, the files are copied to the subdirectory *Abero* in the Windows directory. The *Whvlxd* hacker's utility is then used to

start a clean *MIRC32.exe*. The utility modifies the registry to run itself at every boot, and according to its INI file runs *mIRC32.exe* in a hidden window.

This version of *MIRC32.exe* exactly matches a clean one: it uses an INI file to connect to the desired channel. This makes detection of MIRC32 a bit more difficult: there is no registry entry for it and it is not modified. Similar to our first example, the MIRC INI file enables a number of actions controlled via IRC. The special *Xscan* and *RPCLsa* hacker's utilities are used for IP scanning rather than a batch file. It is able to scan for a number of vulnerabilities and is supplied with an attacker's dictionary of usernames and passwords.

Example 4.
The dropper is a self-extracting RAR archive originally named *postcard.gif.exe*. It installs a clean *MIRC32.exe* and associated INI files into the System directory and then runs a batch file that creates a registry key to run *MIRC32.exe* and then runs *MIRC32.exe* itself. The INI files only contain commands to join channels with multiple nicks and send messages. The aim is not directly malicious, rather to just create bots in channels.

Example 5.
The file is a self-extracting GZip archive. It installs files to *Program Files\Common Files\System\DLL* and runs a clean *MIRC32.exe* in a hidden window using *Whvlxd* as described above. The difference is that this version of MIRC32 is a full MIRC installation that also sets up the key *HKCR\ChatFile* and others associated with MIRC (just like the MIRC is installed on the affected machine). One needs to clean these entries and check whether they belong to the Trojan or to customer's MIRC32. The INI files provide a BNC interface and bot activity on various channels. Additionally the package installs a backdoor.

Classification

By Self-extracting archive type (SfX). Various types of Zip, GZip, RAR and CAB-based SfX are used to deliver these packages. Some of them are commonly used, while others are used mainly in Trojans. The last ones are always decompressed by antivirus and the files inside are scanned. The common ones are decompressed only if they are suspicious, based on names, size, the number of files inside, and the installation script if antivirus is able to read it.
By installation type. The installation path may vary. Most use unusual installation locations inside the Windows or Recycled directories. Most also modify the registry in order to execute at Windows start, using *HKLM\Software\Windows\CurrentVersion\Run* for example, or files registered as services. More advanced ones appear hardly different from legitimate installations.
By modification of legitimate software and the presence of Trojans in the package. Most packages use modified or packed legitimate components, auxiliary tools and often Trojan executables. Those that have no obviously malicious files inside are more difficult to distinguish.
By functionality. For IRCFlood packages, it is typical to have an IRC client joining channels and being able to execute various commands; some concerning bot activity while others may concern the local affected machine, and can include stealing sensitive information, IP scanning, spreading by worm functionality, lowering security settings, etc. Strictly speaking, Trojan and worm activity cannot be discerned from an IRC script file only, one needs to see the associated batch and executable files that run from the script. Although the command and procedure names

are usually straightforward like “ipscan”, “download”, etc, and the installed FTP and DCC agents act in the same way as legitimately installed ones.

Conclusion

We considered malicious packages as a class of malware. They are easy to create (as they are comprised of components that are easily available), usually short lived, and their distribution is exceedingly variable and temporal. SfX and packed executable unpacking and generic detection is essential to provide an acceptable detection rate.

Further Reading

1) http://www.mirc.com
2) http://iroffer.org
3) http://www.servU.com
4) http://www.sysinternals.com (various utilities)
5) http://www.famatech.com (Radmin home)

Malware Penetration Index (MPI) Proposal for New Virus Metric

Authors: Nicky Pappo and Oren Drori

Commtouch Software Ltd.

1. Abstract

This document presents a prediction model for calculating malware penetration probability during an outbreak. The purpose of this document is to promote discussion and encourage suggestions for improvements to the Malware Penetration Index model proposed and developed by Commtouch Ltd. The MPI model is currently being proposed as a new standardized metric for the email security industry. It is suggested that the MPI model be adopted and further developed in order to become a standard measurement procedure for malware penetration probability.

This document also defines the key drivers of this initiative, the theoretic model for MPI, a definition of a new virus metrics and practical methods for MPI estimation. Finally, it outlines possible real-life applications that would benefit from using the MPI model. Distribution of this memo is unlimited.

2. Introduction

For the past few years, messaging application developers, anti-virus (AV) vendors, and the general email messaging industries have witnessed a significant transition in the way that email-borne malware is being pushed to infected hosts and devices. This can be briefly characterized as piggy-backing the most advanced and sophisticated spamming methods, such as propagating extremely short-lived multiple variants, concurrently. Sometimes these variants are repeated in recurrent waves, which are sent via a continuously rehabilitated army of zombies. This makes it extremely hard to proactively trace and block the attack.

In today's reality, the majority of malware is weighted differently by various anti-virus vendors, described differently by a range of online IT forums and trade analysts, and eventually even the media adopts different definitions or threat levels to the same attack. In most cases, the severity of the malware is determined independently based on its incurred damage. This unfortunately only gives value to one aspect of the attack. According to this, the most affecting malware such as a virus, worm, or Trojan gains some public exposure or resulted in a global panic, while the less affecting malware may be hardly recognized or acknowledged.

While gradually more high-risk malware are propagated as a collection of multiple low-risk small malware it is essential that a complementary metric is defined to help measure the potential malware risks that organizations face on an ongoing basis.

Malware Penetration Index (MPI) was developed to empower the community of IT professionals worldwide, who are responsible for the planning and implementation of protection layers and prevention infrastructure. These defense elements require a reliable, unified, and vendor-independent forecasting procedure that can determine how well their organization is sheltered against infiltration of new malware outbreaks, or in other words what is the chance to be infected and get hurt by a new attack. Thereby, allowing them to preemptively make the necessary adjustments to their defense practices.

3. New Virus Metric – Motivation and Requirements

Malware Penetration Index (MPI) is a metric and an attribute of a virus outbreak. It aims to answer the following questions: What are the chances that an email message carrying a virus will successfully penetrate a user's mailbox? Or, put more simply, what are the chances for a successful virus penetration during a virus attack.

The MPI paradigm proposed by Commtouch provides attributes that are desirable in the context of an industry index:

Methodological and mathematically well-defined: if such an index is introduced to the industry, the methodology behind it will make it easier for users to better understand what can and cannot be determined from this measure. The model's transparency model also invites open debate and scientific criticism.

Quantified – (vs. measures such as "severe attack") provides a comparison scale for virus outbreaks: for example, virus A has a larger or smaller MPI value than virus B.

Unbiased / vendor independent – in theory, any two people should be able to properly calculate MPI for a given virus and reach the same answer (excluding any errors in measurement). The MPI model is objective, and not based on subjective decisions of individual vendors or service providers.

4. Defining MPI

MPI is the probability that a random message that was received during a virus attack will penetrate the anti-virus defense and arrive at an end-user's mailbox.

5. Driving Factors

Which factors effect, or should effect, the value of MPI? As MPI measures the bottom line success of virus messages to penetrate to the end-user mailboxes, its value is affected both by parameters that measure the distribution of the virus, as well as parameters that relate to the level of defenses against this virus that were deployed in the world (or, at least in the target population).

	Driving Factor	Impact on MPI Value
Attack parameters	Intensity	↑
	Shape of distribution curve	Right tail distribution (most volume distributed early): ↑ Left tail distribution (most volume distributed late): ↓
	Attack speed	↑
Anti-virus Parameters	Responsiveness of AVs	↑
	Number of effective AVs	↑
	Identity of effective AVs	Large market share - ↑ Smaller vendors - ↓

Theoretic Model for MPI

1. Semantics

The following semantics are used in the MPI model:

Variable	Explanation
I(t)	Intensity Level = the virus proportion (%) of the total traffic, at time *t*
M(t)	Miss Rate = proportion of the unprotected population at time *t*
P(t)	Penetration Rate = at time *t,* the probability of (random) received message to be a successfully penetrating virus (i.e. the message provides for two criteria: • it carries the virus, • the recipient is an unprotected user
MPI(virus attack)	Malware Penetration Index. Define: $MPI := \frac{1}{T}\int_t P(t)$
T	Attack length - i.e. $t \in \{0,1,...T\}$
N	Number of available anti-virus engines i.e. $AV_n \in \{AV_1, AV_2, ...AV_N\}$

Table 1 – Model Semantics

2. Calculating Miss Rate at Time t

Before formalizing the *MPI* derivation, following is an explanation of the logic used to formulate the calculation, geometrically.

The Miss-Rate function, *M(t)* is the percentage of users that are unprotected. *M(t)* typically has a descending-step shape: as anti-virus signatures are released, less people are unprotected.

With each signature release, an entire population of users (that use this anti-virus solution) change their status to *protected*, thus resulting in the descending step shape as the time from virus release increases.

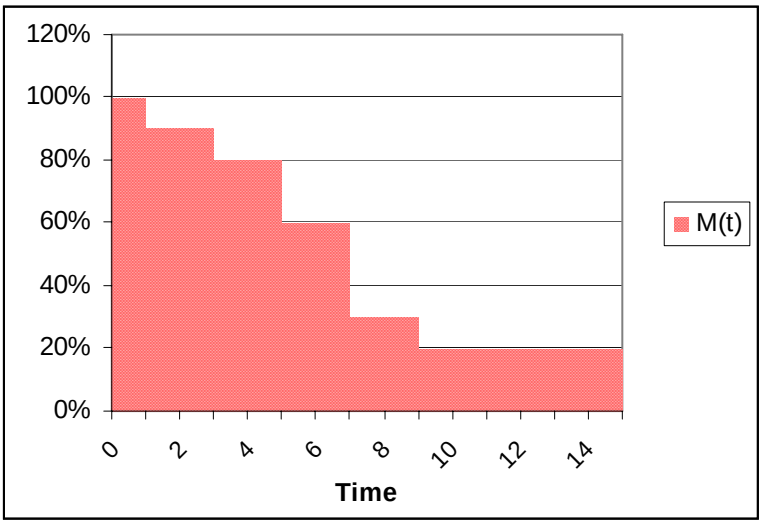


Figure 1 - Miss Rate

The required data for calculating *M(t)* include:

- Determining which anti-virus products provide protection against the virus at time *t*
- The relative market share(s) of the various anti-virus product(s)

To calculate this factor, define:

- $AV_n(t) \in \{1,0\}$ - $AV_n(t)$, which answers the following question:

  Does anti-virus engine *n*, provide protection against the virus?

  The answer to this question is time-dependent, and it can be assigned a value of zero (0) if the answer is *no*, or a value of one (1), if the answer is *yes*.

- S_n - The market share of anti-virus engine *n*. This means the proportion of world users that use this specific anti-virus product.

Hence – aggregating the Miss-Rate across anti-virus engines yields the following calculation:

$$M(t) = 1 - \sum_{1}^{N} S_n AV_n(t)$$

3. Calculating Penetration Rate and MPI

Penetration rate at time *t* is the product of the miss rate which has been calculated in the previous step and the intensity (which is the prevalence of the virus, in percentage points). This yields the following calculation:

$$P(t) = I(t) \cdot M(t) = I(t)\left(1 - \sum_{1}^{N} S_n AV_n(t)\right)$$

An example of a typical penetration rate graph is shown below. It is calculated by multiplying the miss-rate *M(t)* graph and the intensity *I(t)* graph.

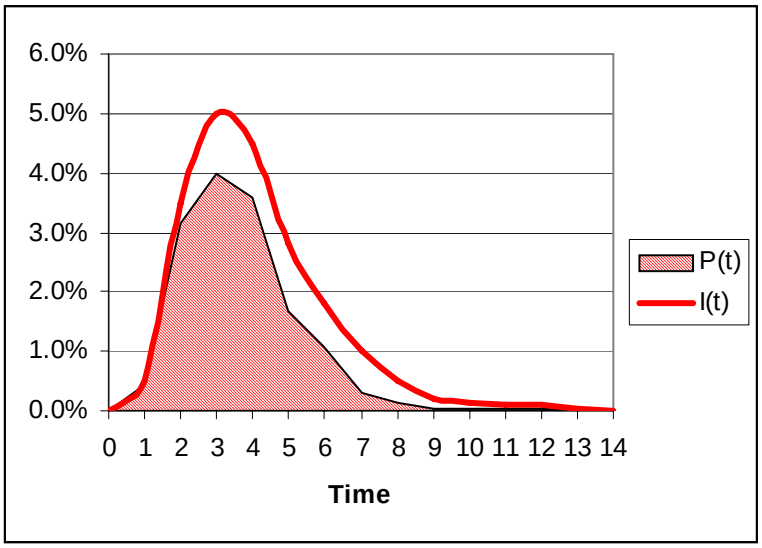


Figure 2 - Intensity and Penetration Potential

MPI is the probability that a single (random) message that was sent at any time during an attack will become a *penetrating message*. In order to calculate this probability, it is necessary to calculate the proportion of the messages that answer the penetration criteria out of the 100% of the messages that were sent during the attack:

Therefore: $MPI = \frac{1}{T}\int_{t} P(t) = \int_{0}^{T} I(t) \cdot M(t)$

Note: Division by *T* is required in order to normalize the index to $(0,1)$ range (between 0 and 100%).

MPI Estimation

In order to properly estimate the MPI for a message, the following procedure is followed:

- Approximating the $AV_n(t)$
- Approximating the S_n
- Calculating the Approximated Value for $M(t)$
- Estimating $I(t)$
- Estimating Penetration Rate (at time t) and MPI

1. Approximating AVn(t):

Approximating $AV_n(t)$ requires:

A lab that consists of the various anti-virus products that are currently available, and is updated throughout the attack. Independent labs like AV-test.org could be selected for this task. Such a lab might not cover every anti-virus engine on the market, but would still provide a good approximation.

A methodology to test the different anti-virus engines at time t – repeatedly until the attack is over. For example, Commtouch labs have built an automated mechanism that performs this action and then repeats it, once it is triggered.

More importantly, a challenge – the above test must be done since the beginning of the attack. A good zero-hour detection mechanism (signature & update independent) must be used for the trigger.

2. Approximating Sn

Approximating the market share of the anti-virus engines can be difficult for a number of reasons:

First, the definition of *Market share* is not trivial because experts use different definitions of the market. In the MPI context the definition should be conclusive and include software solutions as well as hardware solutions, desktop as well as gateway products, consumer as well as business solutions. Typically, available analysts' estimations are partial.

Second, some users are not buying a pure anti-virus product, but rather a security package that includes anti-virus protection. Anti-virus market research usually disregards these users.

Finally, market research usually measure market-share by revenues, while MPI measurement requires market-share by number of units.

In the following example, the approximate market share value (Sn) has been calculated by using a revenue breakdown as published by IDC.

Notes and reservations:

- For simplicity market-share of products that were not mentioned by IDC, was considered as 0.1% for each.
- This approximation will serve reasonably on a global level (MPI as a proportion of infected and penetrating emails out of world traffic), but should be modified

if one wishes to calculate an MPI for a particular market segment, such as business users or consumers etc.

- Inclusion of integrated products in IDC's report is very partial, which create a certain bias.

3. Calculating the Approximated Value for M(t)

Using the approximated $AV_n(t)$ and S_n, you should have the required information to calculate the $M(t)$ variable. To do this, consider a 4-hour attack, approximated market-share data and protection-indications across attack-timeline (provided by lab e.g. AV-test).

The following example illustrates the actual calculation of Miss-Rate at various times - $M(t)$:

	S=share	AV(11:00)	SxAV(11:00)	12:00	SxAV(12:00)	13:00	SxAV(13:00)	14:00	SxAV(14:00)
Symantec	67.6%	0	0.0%	0	0.0%	0	0.0%	1	67.6%
McAfee	12.8%	0	0.0%	0	0.0%	1	12.8%	1	12.8%
Trend Micro	8.1%	0	0.0%	1	8.1%	1	8.1%	1	8.1%
Panda	1.8%	1	1.8%	1	1.8%	1	1.8%	1	1.8%
F-Secure	0.4%	0	0.0%	1	0.4%	1	0.4%	1	0.4%
eTrust-INO	0.2%	0	0.0%	0	0.0%	0	0.0%	0	0.0%
eTrust-VET	0.2%	0	0.0%	0	0.0%	0	0.0%	0	0.0%
@Proventia-V	0.1%	1	0.1%	1	0.1%	1	0.1%	1	0.1%
AntiVir	0.1%	0	0.0%	1	0.1%	1	0.1%	1	0.1%
eSafe	0.1%	1	0.1%	1	0.1%	1	0.1%	1	0.1%
F-Prot	0.1%	0	0.0%	0	0.0%	0	0.0%	1	0.1%
Kaspersky	0.1%	0	0.0%	0	0.0%	0	0.0%	1	0.1%
⋮			⋮				⋮		
Norman	0.1%	1	0.1%	1	0.1%	1	0.1%	1	0.1%
Sophos	0.1%	0	0.0%	0	0.0%	0	0.0%	1	0.1%
VirusBuster	0.1%	1	0.1%	1	0.1%	1	0.1%	1	0.1%
YY_Spybot	0.1%	0	0.0%	0	0.0%	0	0.0%	0	0.0%
ZZ_RAV	0.1%	0	0.0%	0	0.0%	0	0.0%	0	0.0%
Catch Rate			**2.7%**		**11.4%**		**24.2%**		**92.3%**
Miss Rate			**97.3%**		**88.6%**		**75.8%**		**7.7%**

Figure 3 - example approximation of Market Share S_n and Miss-Rate $M(t)$

4. Estimating I(t)

Intensity has to be estimated based on a *Sample Group*. If the sample group is large enough and unbiased, the estimated $I(t)$ will be an unbiased estimator.

- The larger the sample group, the more accurate the estimation will be.
- The more spread the sample is (geographically, by email market segments etc.), chances for bias are lower.

Define:

$In(t)$ as the number of infected messages in the sample (between time *t* and *t-1)*

$E(t)$ as the size of the email sample between time *t* and *t-1*

E as the size of the total email sample, through the attack, that is: $E := \sum_{1}^{T} E(t)$

$\hat{I}(t)$ as the estimated intensity, calculated as: $\hat{I}(t) = \frac{In(t)}{E(t)}$

5. Estimating Penetration Rate (at time t) and MPI

$\hat{P}(t)$ Estimated penetration rate at time *t* will be calculated as:

$$\hat{P}(t) = \hat{I}(t) \cdot \hat{M}(t) = \frac{In(t)}{E(t)}\left(1 - \sum_{1}^{N} \hat{S}_n A\hat{V}_n(t)\right)$$

$$M\hat{P}I = \frac{1}{T}\sum_{t=1}^{T} \hat{P}(t) = \frac{1}{T}\sum_{t=1}^{T}\left(\hat{I}(t) \cdot \hat{M}(t)\right)$$

Alternatively, we can use the following simplified calculation, and more importantly – the process of data collection:

$$M\hat{P}I = \frac{Total\_Penetration\_Sample}{Totla\_Email\_Sample} = \frac{1}{E}\sum_{t=1}^{T}\left(\hat{In}(t) \cdot \hat{M}(t)\right)$$

The two calculations are identical provided that email traffic is distributed evenly through the attack i.e. $E(t) = E(t')$ for any t, t'. This assumption is more plausible when short attacks are considered, than it is for slower attacks.

Example for estimation:

Time	E(t) - sample size	In(t) - infected sample	M(t) - Miss Rate	Penetration(t) = In(t)xM(t)
1	850234	4662	97.3%	4536.1
2	820335	12469	88.6%	11047.5
3	866001	10409	75.8%	7890.0
4	794003	3777	7.7%	290.8
E (total sample):	3,330,573		**Total Penetration:**	23,765

(Total Penetration) / (Total Sample):	**0.71%**

As: $M\hat{P}I = \frac{Total - Penetration - Sample}{Total - Email - Sample} = \frac{4{,}536.1 + 11{,}047.5 + 7{,}890 + 290.8}{3{,}330{,}573} = 0.71\%$

Applications

1. Direct Conclusions

The above information is sufficient to draw the following conclusions:

a. *MPI* = (by definition) the proportion of emails that were both infected and reached unprotected user. For example: "1 of every XX emails through this attack has infected a user"

b. *MPI* = the probability of a (randomly selected) individual message, sent sometime in the attack, to be a successfully penetrating virus.

c. For attacks A & B that lasted for similar periods of time: if virus attack A has a higher *MPI* value than virus attack B, the chances of getting hit by virus A are greater than virus B.

d. Alternatively, it is possible to multiply the attack's *MPI* by the attack length (hours) and compare between any two attacks.

2. Probability of a Random User Getting Hit with a Virus

MPI can be used for estimating the probability of a random user to receive the virus, however several steps and assumptions are required:

a. The probability has to bring into account the attack duration, and the number of emails received through that period of time.

b. Per received message (through the attack timeframe) the probability of getting hit is the MPI and the probability of not getting hit is 1-MPI. [This is an approximation, ignoring the time of the specific message. If we consider receiving time *t* of each message, the probabilities are *P(t)* and *1-P(t)*]

c. If we assume independence between the different emails received by the same user, and the approximation above (i.e. using *MPI* as hit probability for each message):

   Probability of not getting hit by any message = probability of not getting hit by the first, AND not getting hit by the second etc. That is,
   $\Pr(safe) = (1 - MPI)^X$, where *X* is the number of messages received through the attack.

   Probability of GETTING hit through the attack, would then be:
   $\Pr(hit) = 1 - (1 - MPI)^X$

Example:

Consider the following attack and email usage parameters:

- Corporate user – 2 emails per hour on a week day (about 50 a day)
- The attack took place on Tuesday and lasted 10 hours.
- The calculated MPI was 0.1%

3. Conclusion

Total number of emails – 20, the probability of getting hit with a virus would be:
$\Pr(hit) = 1 - (1 - 0.001)^{20} = 1 - 0.9802 \approx 2\%$

Obviously, the Hit-probability responds dramatically to the number of received emails (intense email users are significantly more exposed):

MPI	0.001	0.001	0.001
Hours	10	10	10
Email/hour	1	3	5
Total emails	10	30	50
Hit	1.0%	3.0%	4.9%

The Hit-probability is also very sensitive to the MPI of the attack, for example:

MPI	0.001	0.002	0.005	0.010
Hours	10	10	10	10
Email/hour	2	2	2	2
Total emails	20	20	20	20
No hit	0.98019	0.96075	0.90461	0.81791
Hit	2.0%	3.9%	9.5%	18.2%

Operating System recovery by Anti Virus software, automatic AV tests.

Alex Polischuk and Arkady Kovtun
Computer Associates

Chapter 1: Introducing concept of system infection.

Clear as the sun that we should prevent infection of files or systems using real time monitor.
But sometimes we need scanning files or systems and cure them as well in case of infection.
Our paper is to explain how it is done and to provide best solutions.
Of course we'll need to write whole library to mention every detail of system infection and cure, so we'll try to mention only the most prevalent.

1.1 Difference between system infection and file infected.

Do you remember the time that Anti Virus catch only viruses?
Now we call them malware (malicious software), because they don't infect files only….

With modern malware maturing more and more, we often see it integrating not only into user's files using parasitic or overwriting methods, but rather integrating themselves into the operating system itself. Moreover, with raised complexity of modern operating systems like Windows 2000/XP/Long-horn (now changed to "Vista") there are more and more places in the system where malware component can hide. It can install itself as legitimate "service" or replace any system service.

Today, curing an infection doesn't merely mean removing virus body and reconstructing the file back to the state it was before infection.

The malware does not need to infect as many files as possible to have chance to run, it can just set a registry key or .INI file entry to be executed each time Windows starts, and keep single copy of itself in hidden place on the system, often with obfuscated name. It can replace or delete system files, create new files, change system settings and settings of various applications, thus make our (Anti Virus) job of curing more complicated.

Actually, malware that infects the system rather than files is now dominant: more and more different kinds of malware use system infection methods instead of file infections or use them both.
According to Virus Bulletin magazine, July 2005, most prevalent malware is Win32/Sober.Worm.
Here the part of prevalence table (Virus Bulletin magazine, July 2005):

Win32/Sober 86.66%
Win32/Mytob 5.51%
Win32/Netsky 3.50%
All others 4.33%

These three Worms at the top of the table do not infect files, but infect system.

1.2 Examples of system infection

Description of Win32/Protoride.S!Worm

Type: Worm
Category: Win32
Also known as Win32/Protoride.S.Worm, W32/Protoride.worm (McAfee), W32.Protoride.Worm (Symantec)

Win32/Protoride.S!Worm is memory-resident worm (written in Microsoft Visual C++) that spreads via network shares and acts as an IRC-controlled backdoor that allows unauthorized access to a victim's machine. It has been distributed as a 59,392-byte, UPX-packed Win32.executable. Many strings within the file are in Spanish.

Method of Infection

When executed, Protoride.S attempts to copy itself as WINMNGR.EXE to the following directories (should they exist on an affected machine):
\Documents and Settings\All Users\Start Menu\Programs\StartUp\
\WINDOWS\Start Menu\Programs\StartUp\
\WIN98\Start Menu\Programs\StartUp\
\WINME\Start Menu\Programs\StartUp\
\WIN95\Start Menu\Programs\StartUp\
\WINDOWS.000\StartMenu\Programs\StartUp\
\Documents and Settings\All Users\Menu Iniciar\Programas\Iniciar\
\WINDOWS\Menu Iniciar\Programas\Iniciar\
\WIN98\Menu Iniciar\Programas\Iniciar\
\WINME\Menu Iniciar\Programas\Iniciar\
\WIN95\Menu Iniciar\Programas\Iniciar\
\WINDOWS.000\Menu Iniciar\Programas\Iniciar\
\WIN98\Men Inicio\Programas\Inicio\
\WINME\Men Inicio\Programas\Inicio\
\WIN95\Men Inicio\Programas\Inicio\
\WINDOWS.000\Men Inicio\Programas\Inicio\
\Documents and Settings\All Users\Kynnist-valikko\Ohjelmat\Kynnistys\
\WINDOWS\Kynnist-valikko\Ohjelmat\Kynnistys\
\WIN98\Kynnist-valikko\Ohjelmat\Kynnistys\
\WINME\Kynnist-valikko\Ohjelmat\Kynnistys\
\WIN95\Kynnist-valikko\Ohjelmat\Kynnistys\
\Documents and Settings\All Users\Menu Dmarrer\Programmes\Dmarrage\
\WINDOWS\Menu Dmarrer\Programmes\Dmarrage\
\WIN98\Menu Dmarrer\Programmes\Dmarrage\
\WINME\Menu Dmarrer\Programmes\Dmarrage\
\WIN95\Menu Dmarrer\Programmes\Dmarrage\
\Documents and Settings\All Users\Menuen Start\Programmer\Start\
\WINDOWS\Menuen Start\Programmer\Start\
\WIN98\Menuen Start\Programmer\Start\
\WINME\Menuen Start\Programmer\Start\
\WIN95\Menuen Start\Programmer\Start\
\WIN98\Menu Start\Programma's\Opstarten\
\WINME\Menu Start\Programma's\Opstarten\
\WIN95\Menu Start\Programma's\Opstarten\
\WINDOWS\Start Menu\Programlar\BASLANGI
\WIN98\Start Menu\Programlar\BASLANGI
\WINME\Start Menu\Programlar\BASLANGI

It then copies itself to the %Windows% directory as winmngr.exe and modifies the registry to ensure that this copy is run at each Windows start:
HKLM\Software\Microsoft\Windows\CurrentVersion\Run\Windows Taskbar Manager = Windows%\wintasks.exe"
Note: '%Windows%' is a variable location.
The worm determines the location of the current Windows folder by querying the operating system. The default installation location for the Windows directory for Windows 2000 and NT is C:\Winnt; for
95,98 and ME is C:\Windows; and for XP is C:\Windows.
The worm also creates a mutex "PProtoType_v2:Mutex:sh1t-i_h4v3-n0-m3rcy:<random string>:<random string>:<random string>:(<random number>)" in order to ensure that only one copy of the worm runs at a time.

Method of Distribution
Via Network Shares
The worm attempts to copy itself to unprotected network shares.
If the share is protected by a username and password, the worm attempts to connect using username and password that belongs to current user.

Payload
IRC-controlled Backdoor Functionality
Protoride attempts to connect to a particular IRC server and creates an "invite only" IRC channel.
It then sends messages to any users who have access to the created channel.
Users with access to this channel can see a list of commands (in Spanish), and perform them on victim's machine.
These commands include:
Download files
Execute files
Upload files
List and kill processes
Enumerate dial-up accounts
Hide/unhide program windows
List currently running processes
List current TCP connections
Perform Denial of Service (DoS) attacks
Perform UDP flood attacks (Denial of Service)
Provide system information
Scan IP addresses
Steal passwords

Additional Information
The executable WINMNGR.EXE, contains the following file properties:
Comments: Creado Orgullosamente en Argentina - Made In Argentina
CompanyName: BeyonD aDvanceD TechNoloGies
FileDescription: ProtoType v2.3.0 build 500
FileVersion: 2, 3, 0, 0
InternalName: Pty2Ride
LegalCopyright: Copyright BeyonD TechNoloGieS 2003
LegalTrademarks: BeyonD enGine
OriginalFilename: Rd2.exe

Description of Win32.DlToon!Trojan

Type: Trojan
Category: Win32
Also known as Win32.DlNati.A, TROJ_DLOADER.DZ (Trend),
W32/Dltoon.B.Trojan, Win32.DlToon.C,
Win32/DlToon.C.Trojan, Win32.DlToon.D, Win32.DlToon.E,
Win32/DlToon.E.Trojan, Win32.DlToon.F, Win32.DlToon.F,
Win32/DlToon.F.Trojan, Win32/DlToon.F.Trojan,
Win32/DlToon.F1.Trojan, Win32.DlToon.G,
Win32/DlToon.G.Worm, Win32.DlToon.H,
Win32/DlToon.H.Trojan, Win32.DlToon.I, Win32.DlToon.J,
Win32.DlToon.K, Win32.DlToon.L, Win32.DlToon.M,
Win32/DlToon.Trojan, Win32/DlToons.D.Trojan,
Downloader-DZ (McAfee), Downloader.Tooncom (Symantec),
TROJ_TOONCOM.H (Trend),
Win32/Tooncom.H.Downloader.Trojan, TROJ_TOONCOM.I
(Trend), Downloader.Trojan (Symantec),
TrojanDownloader.Win32.Tooncom.b (Kaspersky),
TrojanDownloader.Win32.Tooncom.c (Kaspersky),
TrojanDownloader.Win32.Tooncom.e (Kaspersky),
TrojanDownloader.Win32.Tooncom.f (Kaspersky),
TrtojanDownloader.Win32.Tooncom.f (Kaspersky),
TrojanDownloader.Win32.Tooncom.h (Kaspersky),
TrojanDownloader.Win32.Tooncom.i (Kaspersky),
TrojanDownloader.Win32.Tooncom.r (Kaspersky)

Win32.DlToon is an increasingly large family of functionally-similar downloading trojans.
The main purpose of the DlToon trojans is for advertising, and generating more hits on particular pornographic and search web sites.
In the process, they often modify user settings, like the IE start page, without the user's permission. They are often installed through exploiting security vulnerabilities in Internet Explorer.
For example, the following is a description of one such variant that has been observed in the field, Win32.DlToon.F (although much of this description is relevant to the other known variants):
Method of Installation:
DlToon generally consists of three files:
loader.exe
iedll.exe
DNSErr.dll
The installation is circular - loader.exe downloads and runs iedll.exe, and iedll.exe in turn downloads the latest loader.exe. Iedll.exe also downloads DNSErr.dll, and stores it in the Windows directory. From reports from the wild, it appears that these components of DlToon are often updated or modified by the trojan's writer. This circular downloading process ensures that the trojan writer's latest variant is
always installed on compromised machines, as these trojans are always checking for and downloading the latest versions of themselves.
Note: Users inflicted with DlToon variants may notice that
DlToon is detected and removed by their CA antivirus solution, but will then be reported as an infection again a short time later. This is caused by the circular downloading process outlined above, and the constant updating of the trojan. If you are experiencing this issue, or similar, we recommend that you search your machine (by selecting Start | Search | For files and folders... from the taskbar) for the files: loader.exe, iedll.exe and DNSErr.dll. In all the variants we have seen prior to the time of publishing this description, the filenames have remained static. If you find these files, please either contact your relevant support group (http://www3.ca.com/support/) and/or submit the files to our antivirus research teams for analysis. For more information on submitting virus samples, please view the Submitting Virus Samples instructions in our Information Center.
As these trojans often gain access to systems using exploits of known Windows vulnerabilities, we highly recommend that users regularly visit http://www.windowsupdate.com to ensure that their machine is always patched against the latest discovered critical vulnerabilities.

Payload
Changes System Settings
The trojan tries to to connect to a particular URL via HTTP and receives instructions to create links in the user's Internet Explorer favorites directory:
For example: in "C:\Documents and Settings\Administrator\Favorites" the following links may be added:

1. ~ Fully categories porn database. Enjoy!
2. ~ New Porn Pics everyday
3. ~ Series Hardcore Pic Sets and Movies

The trojan may modify the following registry entries in order to change the user's default Internet Explorer homepage and/or default start page, by associating them with the URL that the trojan's writer wishes the user to visit:
HKCU\Software\Microsoft\Internet Explorer\Main\Search Bar
HKCU\Software\Microsoft\Internet Explorer\Main\Search Page
HKCU\Software\Microsoft\Internet Explorer\Main\Start Page
HKCU\Software\Microsoft\Internet Explorer\SearchURL
DlToon.F then adds the following values to the registry:
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\loader = current date for instance "17.11.03"
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\loader2 = "1"
HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\loaderGuid = "c8b96a3d-adb5-4db0-a20f-6ecc399c4298"
DlToon.F also modifies the Hosts file (on XP, 2000 and NT systems the hosts file is located at %System%\drivers\etc\hosts: on 9x systems the hosts file is located at %Windows%\hosts). This file contains the mappings of IP addresses to host names. It deletes all existing entries (except localhost) and then maps several adult sites so that they are all redirected to a site that the trojan's writer wishes the user to visit.

Description of Win32/Fantador.E!Trojan
Type: Trojan
Category: Win32
Also known as BackDoor-BCB (McAfee), Backdoor/Fantador.E, Trojan.Riler (Symantec), TROJ_RILER.B (Trend), Troj/Riler-B, Trojan.Win32.Riler.b (Kaspersky)

Win32.Fantador.E is a backdoor trojan that allows unauthorized access to an affected machine.
When executed, Fantador.E creates the following files in the %System% directory:
SPORDER.DLL - this file is clean and not detected by CA Antivirus solutions
WINMEDL.DLL - this is a clean text file that contains the encrypted string: "uygurman.vicp.net" and is not detected by CA Antivirus solutions
SynUSB.dll
WinSSi.exe
Note: '%System%' is a variable location. The backdoor determines the location of the current System folder by querying the operating system. The default installation location for the System directory for Windows 2000 and NT is C:\Winnt\System32; for 95,98 and ME is C:\Windows\System; and for XP is C:\Windows\System32.
The trojan modifies the registry to run each time Windows starts:
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\SynUSB Manager = "rundll32.exe SynUSB.dll,RunDll32"
The backdoor tries to connect to the domain:
"uygurman.vicp.net" and waits for commands from its controller.

Description of Win32/Elkong.D!Trojan
Type: Trojan
Category: Win32
Also known as Win32/Elkong.D, Win32/Elkong.D.DLL.Trojan, TROJ_KEYLOG.AA (Trend), W32/Ovnod.A@pws (F-Secure),
Keylog-Stawin (McAFee)

Win32.Elkong.D is a keylogging trojan.
Method of Installation When executed, Elkong.D copies itself to the %Windows%
directory using the same filename that it was originally executed from, and modifies the registry to ensure that this copy is run at each Windows start:
HKCU\Software\Microsoft\Windows\CurrentVersion\Run\OLE = %Windows%\<Trojan file name>
Note: '%Windows%' is a variable location.
The trojan determines the location of the current Windows folder by querying the operating system. The default installation location for the Windows directory for Windows 2000 and NT is
C:\Winnt; for 95,98 and ME is C:\Windows; and for XP is C:\Windows.
Elkong.D also drops a DLL to the %Windows% directory as HookerDLL.DLL.

Payload
Keylogs/Steals Sensitive Information
The Keylogger waits specifically for windows that contain any of the following strings in the title bar before recording the user's input to the file Windows%\kgn.txt.
1MDC
1mdc
Access
Bank
bank
Bank of Montreal
bank of montreal
Bank West
bankwest
BankWest
bendigo
Bendigo
BMO
bmo
CIBC
cibc
Citibank
commbank
Commonwealth
e-Bendigo
e-bendigo
e-bullion
e-Bullion
e-gold
EVOcash
EVOCash
evocash
goldgrams
goldmoney
GoldMoney
HyperWallet

Elkong.D e-mails the collected keystrokes contained in this file to a particular e-mail address using its own SMTP engine.
Note: This e-mail is not normally displayed to users. Text below the dotted line is displayed for example only.

Chapter 2: Recovering the system

2.1 System infection signs
As we can see from the examples above we could easily define most common places in the Windows system that may be affected by malware.

File system of Windows has free access to common places (folders). Most of those folders are in the common execution path thus malware could be easily executed from registry or from other places. Here are few examples of common folders in Windows that malware typically use:
The boot folder: "C:\"
The Windows folder: "C:\WinNT" or "C:\Windows"
The system folder: "C:\Windows\system32"
And some more:
C:\Documents and Settings\Administrator\Start Menu
C:\Documents and Settings\Administrator\Desktop
C:\Documents and Settings\Administrator\Local Settings\Temp
Favorites folder
Program Files folder
Documents and Settings folder
System Volume Information folder
Recycle Bin folder
Etc.

Few files in Windows 9X are frequently modified by various malware in order to be executed each time Windows starts:
Autoexec.bat
Config.sys
Win.ini
System.ini

The registry may also be an easy target;

Malware love adding registry values to ensure that they run each time Windows starts:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunServices
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services
HKEY_LOCAL_MACHINE could be replaced by *HKEY_CURRENT_USER* for currently login user.

Default value of file execution in registry, for example:
"*HKCR\exefile\shell\open\command\(Default)"* should be "%1" %*
Modify this default value of the command associated with "exefile" will run malware file each time any executable will be called:
HKEY_CLASSES_ROOT \exefile\shell\open\command\(Default) = "<filename>"%1" %"*
Same technique used by malware with various file types in registry, for example:
HKEY_CLASSES_ROOT\VBSFile\Shell\Open\Command
HKEY_CLASSES_ROOT\txtfile\shell\open\command
Same for HTML files Etc.

The start page of Internet Explorer is also one of the favorite system modification places:

Startpage is a large family of trojans that are used to change a user's Internet Explorer homepage and default search page. Generally, these trojans accomplish this by making changes to the registry and the hosts file.
These trojans have been seen in the wild and used by businesses with unethical marketing practices in order to increase the flow of traffic to their web sites.
These trojans often set the following registry entries to point to a site of the trojan writer's choice, redirecting the user every time the default search or homepage is accessed.
*HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explor*er
Values:
Search
SearchAssistant
CustomizeSearch
HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer\Main
Values:
Search Page
Default_Search_URL
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer
Values*:*
Search
SearchURL
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Search
Values
SearchAssistant
CustomizeSearch
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main
Values:
Search Page
Default_Search_URL
Search Bar
Default_Page_URL
Start Page

Additional Instructions for Recovering from a Startpage Infection

Some recent variants of Win32.Startpage exist as a DLL, possibly called "ctrlpan.dll" or "MSCONFD.DLL". This DLL will be loaded through one of
the following registry values:
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\Control = "rundll32.exe %System%\ctrlpan.dll,Restore ControlPanel"
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs = "ctrlpan.dll"
The actual DLL file name may vary.
Startpage changes a user's default Internet Explorer homepage and/or default search page by making changes to the registry. While CA Antivirus solutions will remove a Startpage infection, they will not restore a user's individual Internet Explorer settings to their pre-infection state (as Internet Explorer settings may vary from user to user).
To run any DLL file on Windows restart, the malware may change the following registry value:
HKLM\\Software\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon\\Notify" "DLLName" value = "virus.dll"
The list of registry changes may continue on and on…making Anti Virus job more complicated.

BHO's- Browser Helper Objects use the following registry folder to run their DLLs:
"HKLM\\Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Browser Helper Objects"
The HOSTS file usually found in *%Windows%\hosts* or *%System%\drivers\etc\hosts* folder of Windows.
It maybe modified by malware in order to block access to various sites, computer security sites etc. This way the affected machine will not be able to update its anti virus or view security related site for information.
For instance some of latest Win32/Agobot.Worm variants add the following lines to the *hosts* file:
127.0.0.1 www.ca.com
127.0.0.1 ca.com
127.0.0.1 mast.mcafee.com
127.0.0.1 my-etrust.com
127.0.0.1 www.my-etrust.com
These lines will cause the domains www.ca.com, ca.com, etc. to resolve to the local host, effectively denying access.
DLL is another method for malware to easily add itself to the system. Used by the system makes it "legitimate" and transparent. The latest 'Injection' method makes it even more transparent as malware inject itself to a system process in memory achieving system rights and becoming one with it.

Components of malware are files that malware may create or download.
These may be files of any kind and format some times even clean (not malware) utilities or pictures.
Some of those files defined by our researchers as malware components and should be removed from the system along with original malware file.

Others
Actually there is no limit for infection singes and methods; this paper cannot describe them all.

2.2 System cure

The System Cure was designed to enable complete cure for the infected system, by restoring all of the affected objects. It can perform different tasks depending on the OS.
The infection signs described in previous paragraph will be removed. Specific routines will be provided in Anti Virus update in order to kill malicious processes, delete infected files, remove or change registry values etc.
For all file manipulation actions, it may happen that the file is being used by the system, and therefore cannot be accessed (deleted, renamed). In such a case a system restart is requited. When windows restart, the file manipulation will complete.
It is not always possible to recover the malware-violated system, for instance AV product can't restore deleted files if these files were completely deleted.
But, Anti Virus must do its best to rescue the system, it must restore all system settings at least to sensible default values.
First and most important rule should be same as first rule of medicine: Anti Virus must not harm the system.

Fortunately, malware detection not always means that the system is infected – the file can be there, but nobody executed it.
We need to be sure that the system is infected – this is a must and this is why:
Curing a clean system is not only a lost of time and resources, it could also harm the system causing lost of data.
At the other hand, what will happen if we remove the file on infected system (simple delete)?
All the changes of the system will still remain, some of them will be harmless without the file and some will still perform their malicious tasks.

*2.3 e*Trust Standalone Cleaning Utilities

The CA Security Advisory Team provides a range of standalone cleaning utilities and useful tools for the most widespread and dangerous viruses.
Generally, if you're using the current engine and signatures for eTrust InoculateIT 6.0, eTrust Antivirus 6.0, eTrust Antivirus 7.0, Vet, or EZ Antivirus, and have enabled the System Cure/Clean feature, you will not need these tools or utilities.
CA Antivirus solutions are a complete package, and contain advanced technology that provides you with all the virus protection you'll ever need, including the ability to clean up if things go awry.
These tools and utilities may be especially useful for those who either do not use CA Antivirus solutions, or who may be using products based on older technology that does not support system cleaning.
You're always welcome to download and use these utilities:
http://www3.ca.com/securityadvisor/newsinfo/collateral.aspx?CID=40387

2.3 Automatic system cure

We notice that virus writers often use same techniques to violate the system. Most malware modifies the registry in order to run every time the Windows starts.
We use automatic system cure also called by us as generic system cure.
Generic system cure needs no special routine (written by researcher) on specific peace of malware, it automatically will executed on scanner detection (if Anti Virus configured to cure malware).
First generic system cure will test the system according to known infection signs (some of them are described above) then it will remove them and delete the file.

Chapter 3: System cure quality testing (our lab + virtue)

Needless to say that quality testing is always very important. It becomes vital changing the system configuration.
To help with this "holy" task, we created various tools of integrity checking and monitoring the system.
Created by our researchers together with QA people, those tools should consider any change of the system and after system cure finish its work - the system should return to the stage that it was just before the infection. As we already know, this is not always possible, so our automatic tools are created to decide when ever system was modified to reasonable default values.

Alex Polischuk – Senior Research Engineer, Computer Associates
– Alex.Polischuk@ca.com

Arkady Kovtun – Research Engineer, Computer Associates
– Arkady.Kovtun@ca.com

Computer Associates

References:

1. Virus Bulletin Magazine various editions
http://www.virusbtn.com/

2. Computer Associates security advisor
http://www3.ca.com/securityadvisor/

3. Virus Information Center of Computer Associates
http://www3.ca.com/securityadvisor/virusinfo/

Overview of Australian Law Enforcement Responses to Cybercrime

Gregor Urbas and Rob McCusker
Australian Institute of Criminology

About Author(s)

Dr Gregor Urbas is a Lecturer in Law at the Australian National University (ANU), where he teaches in Criminal Law and Procedure, Evidence and Intellectual Property. He has held research positions in Australia and Europe, and in 2006 is working at the Australian Institute of Criminology (AIC) as High Tech Crime Research Analyst, liaising with the Australian High Tech Crime Centre (AHTCC) hosted by the Australian Federal Police (AFP). He is co-author, with Russell Smith and Peter Grabosky, of the book Cyber Criminals on Trial, Cambridge University Press 2004.

Contact Details: Australian Institute of Criminology, GPO Box 2944, ACT 2601, Australia, phone +61 -2-62609255, fax +61-2-62609201, email Gregor.Urbas@aic.gov.au

Mr Rob McCusker is a Research Analyst in Transnational Crime at the Australian Institute of Criminology. Prior to this he held teaching appointments in transnational crime at a number of universities in England and Wales and was a Visiting Consultant in the USA on e-commerce security. In his current role, he maintains a watching brief on transnational crimes perpetrated by organised crime networks at the global, regional and national levels and provides appropriate policy and operational advice to a number of law enforcement and intelligence agencies within Australia, Hong Kong, the USA and the United Kingdom.

Contact Details: Australian Institute of Criminology, GPO Box 2944, ACT 2601, Australia, phone +61 -2-62609244, fax +61-2-62609293, email Rob.McCusker@aic.gov.au

Keywords

Cybercrime, global threat, law enforcement, legislation, prosecution

Overview of Australian Law Enforcement Responses to Cybercrime

Abstract

Globalisation necessitates the increasing connectivity of the world's computer, banking and financial systems. As most economic and social transactions are processed through computer systems and networks, those using them depend on the efficiency and security of those systems. This creates a vulnerability which criminals can exploit. Since the Internet was designed to be accessible and extraterritorial, it has now become relatively easy to commit crimes remotely and therefore often difficult to identify those responsible and bring them to justice. The range of potential cybercrime threats includes various offences against the confidentiality, integrity and availability of computer data and systems, computer-related traditional crimes, content-related offences, offences relating to infringement of copyright and related rights, and offences concerned with infringements of privacy. Increasingly, phishing, viral and malware attacks are being utilised by organised crime groups to exacerbate the nature, volume and extent of their online activities. For law enforcement agencies there remain difficulties in adjusting from the investigation of traditional crimes with visible suspects and tangible evidence to the investigation of cybercrime with anonymous suspects and intangible evidence. This difficulty is exacerbated by the ability of cyber criminals to operate beyond the confines of national borders. Nonetheless, the legislative and law enforcement response has gained pace over the last five years, with the enactment of the Cybercrime Act 2001 (Cth) in Australia, together with new telecommunications offences. These criminal provisions, and related evidentiary rules and co-operation arrangements under Australian legislation, are in line with the standards of the Council of Europe Convention on Cybercrime which came into force in 2004. Recent national and international law enforcement actions led by the Australian High Tech Crime Centre, particularly in relation to child pornography, hacking and website phishing, have yielded positive results.

Introduction

It is arguable that the internet, with its inherent characteristics of extra-territoriality, speed and anonymity, might have been designed for the perpetration of cybercrime rather than as an enhanced communications platform. It has certainly been fully utilised by business organisations and consumers alike across the globe. The benefits of the uninhibited flow of information, products and currency have, however, inevitably attracted the interest of criminal groups and deviant individuals. They appear to have gained a significant advantage over law enforcement agencies and government bodies in terms of secreting the fact and evidence of their misdeeds. However, with each passing day the response of authorities to cyber crime is becoming more sophisticated and their interventions more successful. The gap between investigator and malfeasant will always remain but, in some areas of criminality at least, that gap is narrowing.

Globalisation and Computer Networks

The process of globalisation has been the facilitator of an unprecedented rise in trade within and between developed and transitional economies alike. In order to achieve this

revolutionary growth and maintain the resultant 24/7 digital economy it has been deemed necessary for there to be an ever-increasing connectivity of the world's computer, banking and financial systems. The current digital climate has provided business organisations with the opportunity for rapid and profitable global product placement opportunities. Unfortunately, while many such organisations have placed high value upon the ability to trade on this basis, few have given serious consideration to the dangers and responsibilities of so doing. Security of corporate computer networks has historically been secondary to the functionality of those networks. Security has tended to be implemented in terms of products such as firewalls and virus detection software rather than through the adoption of a holistic, proactive and overarching security process. The obvious disadvantage with this approach is exacerbated by the differential quality and level of security *per se* that exists within organisations across the globe.

Naturally, the exploitation of these computer networks by businesses would be of short-term utility were it not for a parallel growth in the use of computer networks by the general public. Unfortunately, the public's general gullibility and lack of threat awareness has served only to increase the actual and potential dangers of globalisation. Underpinning many of the problems associated with the globalisation process is the ever-increasing dependency being placed upon computer systems to regulate and control much of daily social and business activity. Computer and telecommunication technology have become intrinsic to the social infrastructure and computer systems and networks have become integral to the utility of the commercial, private and public sectors of society. The high dependency placed upon computer networks for globalised business processes requires that those networks remain highly efficient, and these two factors combined may create potential vulnerabilities in the event of the computer networks concerned failing for technical reasons or because of the sustained activities of criminal networks. Similar considerations apply to the increasing use being made by public authorities of Internet-based communications and information services, often referred to as e-Government.

Vulnerabilities of the Internet

In 2005 there were reported to be over one billion Internet users worldwide (eTForecasts, 2006) and between 300 and 400 million Internet hosts or websites (ISC, 2005). Formerly reticent consumers have taken to online trading with increasing confidence and aplomb but retain, albeit perhaps in fewer cases than before, the requisite degree of carelessness in relation to information contained in emails and websites to remain constant targets of the criminal fraternity. A key consequence of the development of the Internet has been an increase in 'disintermediation', a process in which organisations and their customers meet less and less in person and more and more frequently within an online context. This in turn has led to distributed digital identities in which people's identities appear in digitised form on a number of disparate computer systems. The vastness of this distributed network offers criminal organisations a number of opportunities to exploit the information contained therein. Indeed, the availability of this rich vein of information has given rise to the creation of 'super-empowered criminals' who exist online and provide criminal organisations with pertinent and supposedly confidential information. In 2004, in "Operation Firewall", United States and Canadian authorities arrested 28 people from six countries in relation to a global organised crime network. Operating under names such

as Shadowcrew, Carderplanet and Darkprofits they ran websites through which they bought and sold credit card information (including some 1.7 million stolen credit card numbers) and false identities (McAfee, 2005).

The Infiltration of Transnational Crime

It should be recognised that the well-documented cases of hacking intrusions or spam attacks which have been deemed annoying rather than critical in impact have been elevated to a completely different level by the emergence of transnational and organised crime networks. By their very nature, such networks are loosely structured, motivated by profit and engage in long-term dividend rich enterprise criminality. They are both proactive (in terms of conceiving new crimes or committing old crimes in new ways) and reactive (in terms of their ability to respond immediately to sudden changes in law enforcement activity or legislative change). They too have taken advantage of the globalisation process with its integration of trade, technology, transportation, communications, information and financial systems. Equally, they have exploited the significant movement of population, the utilisation of digital currency, the growth in attractive markets or sources and the differential and often ineffectual responses by law enforcement agencies to their activities. The Internet is an effective haven for such groups in that it provides them with ample criminal opportunities and allows them to commit offences with relative impunity and anonymity. For risk aware and risk averse criminal networks the Internet also constitutes a low risk environment in which their profits can be filtered rapidly via online bank accounts, in which any audit trail will prove difficult if not impossible to follow and in which detection and prosecution remain extremely difficult to conduct with any significant degree of success.

Indeed, the evidentiary burden faced by law enforcement should not be underestimated. Evidence must be admissible, authentic, complete, reliable and credible. Relocated to the cyber environment, ensuring that such characteristics apply to digital evidence is fraught with added difficulties. Modern computer systems have huge storage capacities and consequently the volume of documentation comprising email traffic, graphic files as well as ordinary data files increases exponentially. Technology changes rapidly and obtaining digital evidence from obsolescent systems can prove difficult. Electronic documents can be replicated and altered more easily than paper documents. Computer information may be designed to change over time such as with email systems which automatically remove dated email content. Electronic data may be stored in shared network folders (making identification of the criminal difficult), may be transmitted rapidly and can be erased, corrupted or modified unwittingly. Investigators may be placed under unreasonable time constraints and the nature of the retrieval, preservation and storage of data may have ramifications for its eventual presentation in court. However, law enforcement agencies have made significant inroads into the use of technology such as CCTV and DNA in capturing ‘ordinary’ criminals. It seems likely that their experience in detecting and prosecuting cybercrimes will advance in similar fashion.

Cybercrime Threats

The functionality of computer systems has always taken precedence over the security of those systems. As increased demands are placed upon computer networks it is inevitable

that criminal entities will discover new ways of infiltrating them. In essence, cyber criminals may use computer networks as a target, tool or facilitator of illegal activity (Smith, Grabosky and Urbas, 2004). At a broad level it is possible to discern a number of generic computer crimes. These include offences against the confidentiality, integrity and availability (CIA) of computer data and systems which might involve hacking, deception (e.g. via "phishing" attacks) and espionage. They might also include 'traditional' crimes such as fraud and forgery perpetrated online, content-related offences such as website defacement and the dissemination of false information and offences relating to the infringement of copyright and related rights including unauthorised reproduction and use of computer programs (Council of Europe, 2004).

Criminals have engaged in a wide range of computer-facilitated activity. These have included extortion in which organisations are threatened with sabotage, denial of service attacks or the theft of sensitive information. An example, which also serves to illustrate the cross-jurisdictional nature of the phenomenon, is the attempt by Romanian hackers in 2003 to extort money using threats to send the details of life-support systems used by a South Pole Research Station to an unnamed third country (FBI, 2003). Reputational damage may also be inflicted upon an organisation through defacement of a corporate website, as an act of politically-motivated "hacktivism" or perhaps as part of an enhanced extortion racket. Fraud has grown rapidly in the cyber context with identity theft from online sources being particularly prevalent. Advance-fee frauds (Nigerian '419' scams) have made the transition from clumsily constructed letters and faxes to relatively sophisticated and widely dispersed internet borne activity. Phishing currently constitutes the best-known form of Internet-based fraud in which an email, typically purporting to emanate from a bank, requests the recipient to supply confidential account details. Pharming, an enhanced version of phishing, in which a link is provided to a false website (which nevertheless appears to be genuine) and "spear phishing", which targets a particular business sector or group of individuals, are all prime examples of social engineering which, in this instance, focuses upon the diversion from syntactic attacks (in which the computer itself is targeted) to semantic attacks (in which the computer user is targeted).

By December 2005, the Sophos Anti-Virus service was identifying and protecting against over 114,000 different viruses, worms, Trojans and other malware (malicious software) (Sophos, 2005). The increase in malware has been directly attributed to the interest being taken by organised criminal enterprises in creating modes of entry into potentially lucrative computer networks. Emerging cyber crime threats are likely to include attacks upon mobile devices such as Personal Digital Assistants (PDAs) and laptops, the use of email to disseminate malware and the exploitation of Wireless (Wi-Fi) computer networks which remain open to remote interception of the information stored upon them (McAfee 2005). It is noteworthy that during 2005, cyber criminals focused their attention on creating software that would, *inter alia*, steal information, note the key strokes made by the user (key logging), turn off anti-virus protection measures or exploit latent vulnerabilities within the computer's existing software. The rise, and use by organised criminals, of bots (robots) and botnets (robot networks) to provide (via viral contamination) effective remote control of computers (zombies) so as to facilitate the targeting or use of such computers in the pursuit of crime has been noted as a significant

trend (McAfee, 2005). Three examples of organised criminal groups who have created malware are Superzonda, HangUp and Shadow Crew. In July 2003, it was noted that Superzonda, for example, was believed to have generated 50 million spam emails per day (Lander and Wearden, 2005). Spyware, which remains secretly on targeted computers and gathers pertinent information, is on the rise. In January 2005, 54.2% of threats were spyware. By November 2005 that proportion had risen to 66.4% (Sophos, 2005).

Legislative and Law Enforcement Response

Australia's federal legal system encompasses State, Territory and Commonwealth jurisdictions, each with its own set of criminal laws – including cybercrime laws. Although there is some overlap and considerable harmonisation with an increasing dominance at the Commonwealth level, this legal complexity continues to be somewhat of an impediment to efficient law enforcement. Nonetheless, a fairly robust and modern set of criminal offences and law enforcement powers is now available, covering most forms of computer misuse.

Substantive Offences

Australian computer offences date from the early 1980s, when new provisions were first enacted to deal with information on government computers, but which have been expanded to cover most forms of illegality committed through or directed at computers (Grabosky and Smith, 1997; Grabosky, Smith and Dempsey, 2001). The evolution of other computer offences largely replicated the experience in Europe and elsewhere, which emerged in a series of waves directed towards privacy concerns, economic crimes, intellectual property protection, illegal and harmful content, criminal procedural law and security law (Sieber, 1998). Specific offences dealing with unauthorised access to computers and data were enacted in response to the growth of "hacking" activity in the late 1980s, including offences under the *Crimes Act 1914* (Cth) protecting Commonwealth computers and data against intrusion (Bronitt and Gani, 2003; Steel, 2002). By the end of the 1990s, the Commonwealth substantially expanded its jurisdictional reach by relying on its legislative powers with respect to telecommunications (Smith, Grabosky and Urbas, 2004). The *Cybercrime Act 2001* (Cth) relocated the main computer offences within the *Criminal Code Act 1995* (Cth), and new offences were included relating to electronic communications such as Internet and email. In particular, these laws criminalise any unauthorised access to or modification of computer data or impairment of electronic communications, where this is caused by means of a telecommunications service, with intent to commit a serious Commonwealth, State or Territory offence (s477.1). This means that a good deal of computer hacking via the Internet, for example, in order to defraud or extort money, now falls within the scope of the *Cybercrime Act* (Urbas and Smith, 2004).

General regulation of the Internet largely falls under the *Telecommunications Act 1997* (Cth), which includes measures that can be used to remove offensive or unclassified material from websites hosted in Australia. Other legislation may also be used to control Internet content. The *Racial Discrimination Act 1975* (Cth), for example, contains provisions according to which material made available to the public, for example through websites, can be declared 'unlawful' and ordered to be removed. However, criminal penalties do not apply under this statute, though there is a range of State and Territory

racial vilification offences that do carry criminal penalties. An example of the application of the *Racial Discrimination Act* to an Australian website containing material questioning the historical occurrence of the Holocaust is the litigation surrounding the "Adelaide Institute" website (Seeto, 2002; Lim, 2002; Taylor, 2001)

The Commonwealth further expanded its regulatory control over the Internet with the enactment of the *Spam Act 2003* (Cth). This legislation does not contain criminal offences as such, but rather provides for civil penalties that allow for the imposition of large fines on persistent senders of unsolicited electronic messages (McCusker, 2005). Finally, the Commonwealth has continued to expand its jurisdiction in relation to the Internet and electronic communications by enacting a suite of new telecommunications offences, including offensive content, child pornography and child abuse material offences (Wong, 2005). Some charges under these provisions have been laid after recent international online child pornography investigations.

In addition to these telecommunications provisions, there have been new terrorism offences enacted over the past few years that may extend to forms of computer misuse approximating "cyber-terrorism" (Urbas, 2005). The definition of 'terrorist act' in s100.1 of the *Criminal Code Act 1995* (Cth) specifically includes acts directed at electronic systems such as information, telecommunications, financial, public utility and transport systems.

Procedural Provisions

These substantive computer-related offences outlined above have been supplemented by a range of procedural provisions that assist law enforcement and give investigating officers special powers in gathering electronic evidence. For example, the recently enacted telecommunications offence provisions impose obligations on Internet service providers (ISPs) and Internet content hosts (ICHs) to alert police about suspected online child pornography and child abuse material (*Criminal Code Act 1995* (Cth), s474.25). Other amendments added by the *Cybercrime Act 2001* (Cth) include provisions for the use of electronic equipment to conduct forensic examinations at warrant premises (*Crimes Act 1914* (Cth), s3K), use of equipment found at warrant premises (s3L), and powers to compel persons with knowledge of passwords or computer security protections to assist investigators (s3LA). Similar provisions are found in the *Customs Act 1901* (Cth) and the *Telecommunications Act 1997* (Cth), the latter including (in s547J) powers specific to the enforcement of warrants relating to suspected breaches of the *Spam Act 2003* (Cth).

In some circumstances, a person who assists others in concealing criminal activity by means of encryption, steganography or destruction of data may be charged as an accomplice. Others who assist in setting up systems for criminal exploitation may be similarly liable. It has been reported that organised crime groups are increasingly turning to online phishing frauds and identity theft (Espiner, 2006), which necessitates enlisting the skills of technologically proficient people such as website designers. The recent indictment of a botnet dealer in the US charges that he advertised the sale of access to the computers under his control to those interested in launching distributed denial of service (DDOS) attacks or undetected spam (Anonymous, 2006).

International Co-operation

In Australia, as elsewhere, there are two basic forms of international co-operation relating to cybercrime: (i) informal investigator-to-investigator assistance, such as where investigators in different jurisdictions share information which is publicly available and thus no special legislative basis is needed; and (ii) formal regimes for mutual assistance and extradition, based on the *Mutual Assistance in Criminal Matters Act 1987* (Cth) and *Extradition Act 1988* (Cth). Formal mutual assistance is invoked when an investigation entails compulsory powers such as the execution of a search warrant in the requested country, or the extradition of a suspect to the requesting country (Cuthbertson, 2001). Bilateral or multilateral agreements on mutual assistance and extradition can expedite the processes, but even so, the need for ministerial or other high level oversight in both the requesting and requested country means that the machinery of formal mutual assistance turns slowly. Australia currently has bilateral mutual assistance treaties with about 20 countries.

Australia is also a member of Interpol, and the Australian Federal Police (AFP) has liaison offices in over thirty countries around the world. Conversely, other nations have law enforcement officers posted to Australia. Assisting in cybercrime investigations is the establishment of a 24/7 network of contacts, enabling nations to alert each other in real time to urgent matters. Australia's contact point for this network is the Australian High Tech Crime Centre (AHTCC), housed at the headquarters of the Australian Federal Police (AFP). These arrangements, which allow for a "fast freeze, slow thaw" process for the preservation of volatile electronic evidence, have been established pursuant to the Council of Europe (COE) *Convention on Cybercrime*. These arrangements are available not only to formal signatories, but also to sympathetic non-signatory states. Although not a signatory to the Convention, Australia has formally signed on to these arrangements.

At the international level, the COE Convention provides a significant achievement in setting minimum requirements for both procedural and substantive cybercrime laws. Provisions relating to evidentiary issues include those dealing with the expedited preservation of stored computer data (Art. 16), expedited preservation and partial disclosure of traffic data (Art. 17): production orders (Art. 18), search and seizure of stored computer data (Art. 19), real-time collection of traffic data (Art. 20) and interception of content data (Art. 21).

Recent Enforcement Actions

The most publicly visible enforcement actions against transnational cybercrime threats in Australia have been directed at online child pornography rings. International cooperation between countries including Australia in combating this illegal trade and the physical and sexual abuse of children that it both depends upon and promotes dates back to the 1990s, highlighted by the arrest of members of the "Orchid Club" in the United States, Canada, Finland and Australia, and the "Operation Cathedral" action against members of the "Wonderland Club" across 14 countries in Europe, North America and Australia. The activities uncovered included both trading of images and real time photography of children in sexually explicit poses, and transmission online in an interactive session with other network members (Grant, David and Grabosky, 1997). More recently, "Operation Falcon" has snared hundreds of traders of online child pornography worldwide, with the

Australian part of this investigation codenamed “Operation Auxin” identifying over 700 suspects in 2004. The activity was centred on a company in Belarus, and police were able to track credit card details of the many customers who had paid to download content (Krone 2005). Most recently still, another worldwide Internet-based child pornography investigation has resulted in 27 persons in the United States, Canada and Australia being charged. The abuse activity included live video streaming of children as young as 18 months being sexually abused, with members of a chat room called “Kiddypics and Kiddyvids” watching in real time (US Department of Justice, 2006).

Other forms of Internet-based crime have also been the targets of successful and multi-jurisdictional cooperative enforcement actions. Australia has been dealing with hackers since the early 1980s (Smith, Grabosky and Urbas, 2004), and some of the intelligence on the activities of Australian members of hacking communities has come from United States and other overseas police services. In a notable and very recent incident, the Australian High Tech Crime Centre has been involved in the investigation and arrest of a Melbourne man in relation to sophisticated botnet attacks on Internet relay chat (IRC) servers in Australia in 2005, with effects also in the United States, Singapore and Austria. On this occasion, the main information came to Australian authorities from the Belgian Federal Computer Crime Unit (AHTCC, 2006).

Another area of increasing prominence is phishing attacks that attempt to trick banking customers into providing personal and financial details to fraudsters, using fake websites and emails designed to appear genuinely connected to real banks and financial institutions. In a particularly brazen example, scam emails purporting to be from the Reserve Bank of Australia (RBA) were sent to recipients requesting them to log onto a website to verify their credit card details. The RBA has taken the unusual measure of posting an example of the emails on its (genuine) website (RBA, 2006). While the source of these emails is yet to be publicly announced, it does appear that increasing numbers of Internet frauds experienced in Australia are connected to organised crime groups based overseas, particularly in Eastern Europe and former parts of the Soviet Union.

Finally, there has been recent enforcement activity directed at copyright and trade mark infringement facilitated by file-sharing and warez-trading networks using the Internet. An example occurred in Australia in 2003, with three operators of a website called “MP3 WMA land” convicted for large-scale distribution of pirated material, two being sentenced to 18-month suspended sentences and the third to 200 hours’ community service (Pearce, 2006). The United States has also been active in pursuing alleged Australian members of software piracy rings, with one suspect having been indicted and pursuing legal challenges to extradition over the past 18 months. This suspect lost an appeal before the Full Federal Court in March 2005 and was refused special leave to appeal to the High Court of Australia in September 2005, and awaits extradition to face charges of conspiracy and criminal copyright infringement in the United States (Hayes, 2006).

Conclusion

Globalisation paved the way for the full utilisation of the internet and the internet facilitated the rapid growth of a digitally enhanced global trade in a myriad of products.

In order to best exploit such developments it has become essential for the computer systems and networks upon which such trade is performed to be linked on a sustained and effective basis. Dependence upon this connectivity has led to potential dangers in the event of a security breach. Criminals have actively sought to exploit the huge swathes of information stored upon and large volumes of financial value transmitted through those networks. The characteristics of speed and flexibility that render the internet of use in a commercial context also facilitate criminal malfeasance. It is entirely possible for criminals to operate clandestinely in cyber space and extremely difficult for law enforcement agencies to detect, extract and present evidence of their misdeeds. However, legislative endeavours and rapidly accruing experience are slowly but surely approaching cyber criminality with ever-increasing confidence and success.

References

Australian High Tech Crime Centre (AHTCC)(2006). International Internet Investigation Nets Arrest, media release of 22.03.06. Accessed 22.03.06 at http://www.ahtcc.gov.au/.

Anonymous (2006). "Botnet" Dealer Indicted in First Prosecution of Its Kind, 23(1) *Computer and Internet Lawyer* 22.

Bronitt, S. and Gani, M. (2003). Shifting Boundaries of Cybercrime: From Computer Hacking to Cyberterrorism, 27 *Criminal Law Journal* 303.

Council of Europe (2004). *Organised Crime Situation Report 2004: Focus on the Threat of Cybercrime*.

Cuthbertson, S. (2001). Mutual Assistance in Criminal Matters: Beyond 2000, 75(5) *Australian Law Journal* 326.

Espiner, T. (2006). Interpol: Give us the tools to fight cybercrime, ZDNet UK article (dated 22.03.06). Accessed 22.03.06 at http://www.zdnet.com.au/news/security/soa/Interpol_Give_us_the_tools_to_fight_cybercrime/0,2000061744,39247247,00.htm.

eTForecasts (2006). Worldwide Internet Users Top 1 Billion in 2005, Says eTForecasts, Tekrati – The Industry Analyst Reporter, research news of 05.01.06. Accessed on 22.03.06 at http://www.tekrati.com/research/News.asp?id=6275.

Federal Bureau of Investigation (FBI)(2003). The case of the Hacked South Pole, media release of 18.07.03. Accessed on 22.03.06 at http://www.fbi.gov/page2/july03/071803backsp.htm.

Grabosky, P.N. & Smith, R.G. (1998). *Crime in the Digital Age: Controlling Telecommunications and Cyberspace Illegalities*, Transaction Publishers/Federation Press, New Brunswick, New Jersey.

Grabosky, P.N., Smith, R.G. & Dempsey, G. (2001). *Electronic Theft: Unlawful Acquisition in Cyberspace*, Cambridge University Press, Cambridge.

Grant, A., David, F. and Grabosky, P.N. (1997). Child Pornography in the Digital Age, 3(4) *Transnational Organized Crime* 171 – 188. Accessed 22.03.06 at http://www.aic.gov.au/publications/chpornography/.

Hayes, S. (2006). Local fights US piracy charges, AustralianIT news story of 10.01.06. Accessed on 22.03.06 at http://australianit.news.com.au/.

Internet Systems Consortium (ISC)(2006). Internet Domain Survey Host Count. Accessed on 22.03.06 at http://www.isc.org/.

Krone, T. (2005). Operation Auxin: The Australian Response to Online Child Exploitation, 67(3) Gazette (Royal Canadian Mounted Police). Accessed 22.03.06 at http://www.aic.gov.au/publications/other/2005-krone.html.

Lander, A & Wearden, G. (2005). Criminals Send Malware Levels Soaring. Accessed 17.03.06 at http://www.zdnet.co.uk/print/?TYPE=story&AT=39207187-390020375t-10000025c.

Lim, Y.F. (2002). Jones v Toben – Racial Discrimination of the Internet, 5(4) *Internet Law Bulletin* 34.

McAfee (2005). Virtual Criminology Report: North American Study into Organized Crime and the Internet.

McCusker, R. (2005). Spam: nuisance or menace, prevention or cure? *Trends and Issues in Crime and Criminal Justice*, No. 294, Australian Institute of Criminology. Accessed 22.03.06 at http://www.aic.gov.au/publications/tandi2/tandi294.html.

Pearce, J. (2003). Aust music pirates sentenced, ZDNet news story of 18.11.03. Accessed on 22.03.06 at http://www.zdnet.com.au/news/business/0,39023166,20281145,00.htm.

Reserve Bank of Australia (RBA)(2006). Warning – Hoax Emails, media release of 24.01.06. Accessed on 22.03.06 at http://www.rba.gov.au/MediaReleases/2006/warning_240106.html.

Seeto, L. (2002). Race hate and the Internet, 50 *Computers and Law* 21.

Smith, R.G., Grabosky, P.N. and Urbas, G. (2004). *Cyber Criminals on Trial*, Cambridge University Press, Cambridge.

Sophos (2005). Sophos Security Threat Management Report.

Steel, A. (2002). Vaguely Going Where No-one Has Gone: The Expansive New Computer Access Offences, 26 *Criminal Law Journal* 72.

Taylor, G. (2001). Casting the Net Too Widely: Racial Hatred on the Internet, 25 *Criminal Law Journal* 260.

Urbas, G. (2000). Public Enforcement of Intellectual Property Rights, *Trends and Issues in Crime and Criminal Justice*, No. 177, Australian Institute of Criminology. Accessed 22.03.06 at http://www.aic.gov.au/publications/tandi/tandi177.html.

Urbas, G. and Smith, R. (2004). Computer Crime Legislation in Australia', 7(2) *Internet Law Bulletin*, 53-56.

US Department of Justice (2006). Prepared Remarks of Attorney General Alberto R. Gonzales at Announcement of Criminal Charges in International, Internet-Based Child Pornography Investigation, March 15, 2006. Accessed 22.03.06 at http://www.usdoj.gov/ag/speeches/2006/ag_speech_060315.html.

Wong, E. (2004). Tough new laws for child pornography and other internet offences 7(7) *Internet Law Bulletin*, 89, 104.

Pharming: a real threat?

David Sancho & François Maillard

Trend Micro, Inc.

About the authors

David Sancho is an Anti-Malware researcher in TrendLabs, part of the Trend Micro worldwide laboratories in Cork, Ireland.He has written and has published a number of research papers on malware tendencies, has been featured on television and radio and participated in customer events where he has presented on business issue and malware related topics.

Contact details: Trend Micro EMEA, Business & Technology Park. Model Farm Rd. Cork, Ireland

Telephone: +353 21 730 7339, email: David_Sancho@trendmicro.com

François Maillard is a software engineer in Trend Micro EMEA support department in Cork, Ireland. He is a Linux enthusiast and in addition to networking skills, he has always liked working on small programming projects (from small Logo programs to image recognition in C++).

Contact details: Trend Micro EMEA, Business & Technology Park. Model Farm Rd. Cork, Ireland

Telephone: +353 21 730 7407, email: Francois_Maillard@trendmicro.ie

Keywords

Pharming, DNS, vulnerability, attack, phishing, exploit, botnet, poisoning, brute force, domain, sniffing, spoofing, malware, antivirus, Denial of Service

Pharming: a real threat?

Abstract

This paper presents a modern-day assessment of Pharming attacks. It provides technical background on how Pharming attackers can subvert the DNS systems to alter legitimate queries and redirect users to different sites. Traditionally, this type of attack has been linked to Phishing, which focuses on financial attacks. The paper explains what other malicious effects Pharming can have and it also describes the current solutions to the problem as well as their effectivity.

Introduction

If the Internet is a building, TCP/IP protocols are the bricks and DNS services are the foundations of the whole structure. Users rely so much on Internet names that the services that provide those names become crucial. If a malicious remote user attacks a web service, it might compromise the machine. On the other hand, if that malicious person attacks the DNS server that points to the organization the outcome can be worse. It can affect all traffic coming from or going to that server and have a devastating effect on the whole network. Pharming is such an attack.

Discussion

We have been hearing about Pharming for some months now and most of us are still wondering how much of a threat it really is. The name is supposed to have started as a pun on the banking fraud called Phishing. Though there have been Pharming incidents reported in the past, it does not seem to be a very popular attack. The purpose of this paper is to give a detailed technical account of the problem. It will also provide some possible solutions and an assessment of the risk of falling victim to it.

The free online encyclopedia “Wikipedia” defines Pharming as follows: Pharming is the exploitation of a vulnerability in the DNS server software that allows a cracker to acquire the Domain name for a site, and to redirect that website's traffic to another web site.

The flexibility of this kind of attack is impressive. A successful DNS cache poisoning attack can change the IP address of a given website and thus redirect all traffic to a different one. In fact, it could not only change websites, but any other Internet service, from FTP to email exchangers. Some possible malicious uses this attack can have (but is not limited to) are:

- **Phishing**: Redirect banking web traffic to fake web pages that are identical in look and feel to the bank’s sites to steal usernames/passwords. This trend started in early 2003 with attacks via email. The attacker would spam emails apparently coming from a bank. In this official-looking notice, they would ask the user to log into the bank’s webpage with some excuse (verify personal data, confirm something...). The trick is that the link does not point you to the bank site, but to a fake website that looks identical. This copy page will ask for the user’s login, which is exactly what the attacker needs to steal the victim’s resources from the real bank. In a similar fashion, redirecting a banking web page’s traffic to a fake website will point the unknowing user to the fake bank page, but this time it is much worse. In the email attack, the user did something wrong: click on a trap link. In the Pharming attack, the user cannot know that he is being redirected to a fake site so the trick is much more likely to succeed.
- **Program distribution**: Redirect all HTML traffic to web pages that install Adware, Spyware or malware. With this, the attackers can install any program they like on each of

the redirected users' systems. This is the most dangerous and effective trick, since the attacker does not really care what domains he has attacked successfully, the result will be the same in any case. In order to avoid any suspicion, the website can even redirect the user to the correct website, once the malware is already in the system. The user would only notice that his favorite website needs to install a new plug-in to work. It is very easy to combine this technique with HOSTS file hijacking if the installed program modifies the host file. With such an action, the malicious program would hardcode the infected system to point to different websites instead of the real ones. This would have the same effect as Phishing if the hijacked domains were banking sites.

- **Antivirus retaliation**: Redirect all antivirus software traffic to fake websites to prevent updating of the products and increase the lifetime of a malware attack. Worms and trojans have done this using a different method. By altering the hosts file, malware can point the antivirus software to different websites so they are always outdated. Using Pharming to accomplish the same goal is mostly the same trick. Note that this strategy can only work in tandem with some of the other attacks, as it is inherently defensive. For example, it can be useful for the attackers to have a Botnet perform Pharming attacks of this nature to increase the Botnet's lifetime.
- **Email Address Gathering**: Redirect all email traffic to a fake server to gather email addresses to sell to spammers. They could set up a server to redirect all email from the stolen domain. It is unimportant whether the server delivers the emails, because they can just log all recipient and sender addresses before delivery and create a valuable email database. This very easy attack can go almost unnoticed if properly done.
- **Stealing Botnets**: Redirect all IRC traffic to a different server to steal zombies from a Botnet. In the last two years, the AV industry has seen a constant rise of a new kind of threat: bot worms. These programs connect to a central server by means of the IRC protocol and await orders from the owner of the so-called Botnet. The potential threat of the attack is directly related to the number of hosts the attacker controls. Pharming could direct the efforts to changing the IP addresses of known existing bot master servers in order to take control of their attached bots. If zombies were the ones to perform the attack, it could be a new propagation method for the Botnet, but this is only hypothetical.

Though these are only a sample of the possible uses of Pharming, they are scary enough to make everybody understand how dangerous this attack could be. In fact, Pharming is a powerful attack. With a bit of imagination it can not only be a mere nuisance, but also a troubling opponent for security vendors.

An attacker can use many tricks to redirect users from a real website to a fake one. They range from local attacks (host file hijacking) to potentially affecting many users (cache poisoning). This article focuses on the higher scale attacks, since the 'Pharmer' can use them globally and therefore, they are much more dangerous.

To get a clear idea of what a Pharming attack is about, it is necessary to know how the DNS system works. Having a basic knowledge of the DNS exchange will allow you to understand how it can be broken.

DNS system mini-primer

The Domain Name Server (DNS) mechanism is the basis of the Internet naming system. DNS allows us to refer to servers in the Internet by a friendly name (www.eicar.com), instead of their IP

address (12.34.56.78), which is the only way machines can communicate with each other. In order to do so, DNS servers hold a distributed database with the domain names and their corresponding IP addresses. When the user attempts to access a given webpage the DNS service will translate the name to the corresponding number, much like searching in a phonebook. The DNS database is very distributed and DNS servers query each other until they locate the correct piece of information.

Only some servers are authoritative for a certain domain. A DNS server is authoritative when it holds the information in its local database. As a real example, take for instance www.eicar.com. The DNS server, which is authoritative for this domain, is 81.3.2.135. This means that all accesses to the page would first have to query this server to obtain the IP address of the page. Figure 1 displays this information flow.

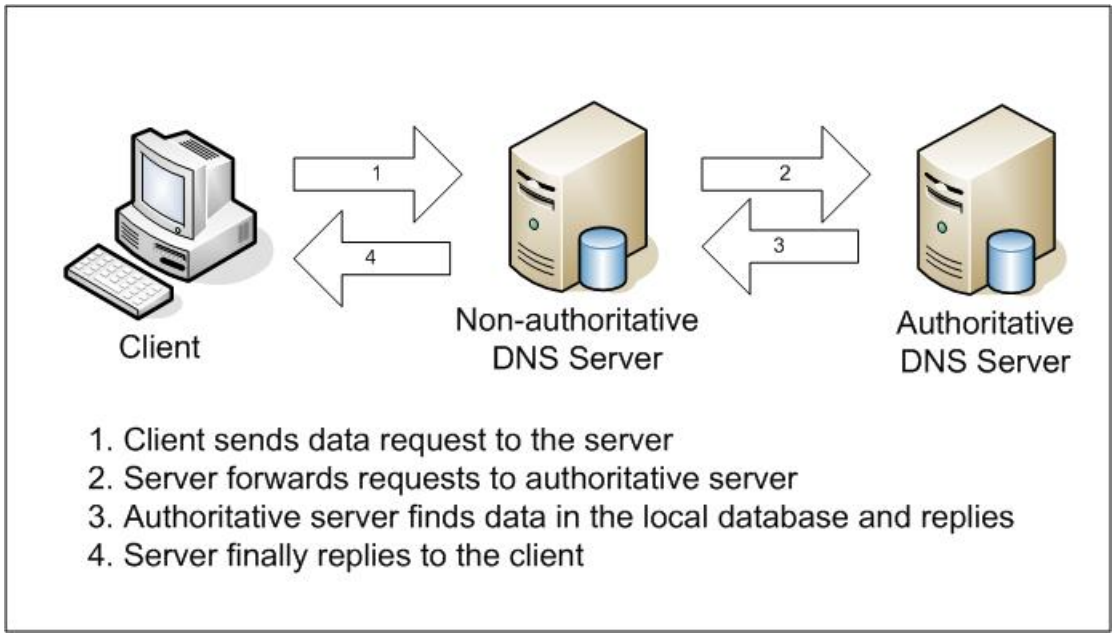


Fig.1: Shows the flow of a normal non-authoritative DNS query.

Based on this DNS system of querying and receiving responses for each domain name, the attacker can use different techniques to redirect traffic from a real website to a fake one. The following strategies will allow the attacker to setup a fake website and receive the stolen traffic, without user intervention.

Hosts file hijacking

Before a Windows client performs an external DNS query, there is a small local database, which can hold pairs of domain name, IP address. This is the HOSTS file, which is located in "c:\windows\system32\drivers\etc". This text file can hold in itself any number of domain name data and Windows will always look there and try to find a match before making a DNS query to the default DNS server. The purpose of this file is to allow system administrators to accelerate domain queries by hardcoding their corresponding IP addresses and save DNS traffic. Any new entry in this file would match the referred web page to a different IP address. In the past, many worms and trojans used this trick successfully to prevent the antivirus software from updating. More recently, they use it to point banking web pages to Phishing sites in order to steal the user's bank credentials.

The attack is very easy to replicate. The attacker only has to add a line in the hosts file with the new name-address pair, such as “www.bank.com 11.22.33.44”. After doing this, the user would connect to the fake address whenever he tries to access the mentioned domain name. In larger-scale attacks, the model would be the following. First, the Pharmer uses some other trick to point the user to a malware website. Since the user typed the address himself, it is likely that he will accept the malware download, thinking it is legitimate. Once the malware becomes active, it can add any number of entries in the hosts file to capture the other web pages and redirect them to different fake sites. With this method, the attacker does not have to care about which domain to target in the attack, as any popular domain would work just the same.

This is a trivial procedure to implement, but it is also very limited in scope. This limits its effectiveness and is very easy to detect by trivial antivirus and Antispyware monitoring tools.

DNS settings modification

Along the same lines as the previous attack, the malicious user or component can also change the configured DNS server in the system to point to a remote malicious server. This would effectively give control of the settings to the attacker. If they do this, they can decide what pages to redirect or even have the power to redirect all domains. Though this is also a local attack, if successfully performed, it can compromise all the outgoing traffic of the affected computer.

New DHCP server

This is also a local attack, but it could compromise the whole network. DHCP servers provide clients with an IP address and some other network parameters, among which is the default DNS server. In this scenario, a malicious local user sets up a new DHCP server providing a remote malicious DNS server to connecting clients. The result is that rebooted clients would point to it and their outgoing traffic would be in the hands of the attacker. This attack has an additional risk, which is that of having two DHCP servers in a given network. The two DHCP servers would compete with each other in sending their responses to new clients.

An additional barrier is the fact that the attacker has to know the exact configuration of the network. If he does not, there is a risk of disabling clients by giving them duplicate IP addresses and the possibility of denying them access to the Internet.

From 2003 onwards, Cisco IOS supports a feature to prevent this. The administrator can select what interface the DHCP server is allowed to use within the switch. Any other DHCP transmission will be ignored. Cisco specifically released this “DHCP Snooping” feature to avoid having rogue DHCP servers in the network.

DNS Spoofing with sniffing

Even though this is a local network attack, it is powerful enough to grant it a spot in the most effective attacks ranking. The attack revolves around capturing DNS server query packets and forging the response. When this operation is successful, the DNS server will think that the fake response coming from the hacker is in fact a real response from a remote DNS server. The DNS server will think that the IP address of the response is legitimate, when in fact it is not. Figure 2 displays a diagram of the attack.

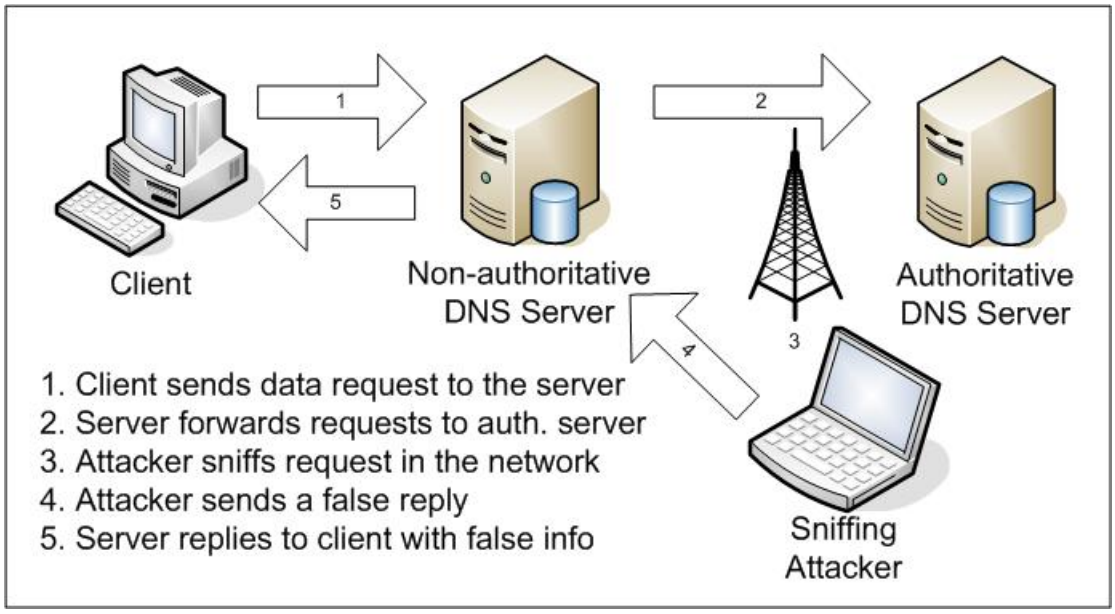


Fig.2: Shows the flow of a non-authoritative DNS request when there is a sniffing attacker in the network. Note how the real reply from the authoritative server arrives later than the local one and is therefore ignored.

To test the feasibility of this threat, we tried to replicate it in our test lab. We created a program that sniffs DNS traffic in the local network from the target DNS server to other external servers. When the program detected a query, it would send a fake response posing as the external DNS server. In our tests, we were able to fool the target server every single time. This is possible because of the fact that the attacker already has all the data he needs in order to correctly spoof the real response. Since the hacker program already has the original request, it is not complicated to build a matching response that the server will accept as good. Of course, this response will point the attacked server to a different IP address and not to the real one. When the real response comes from the remote server, it is already too late, since the local hacker was faster and his response won the race. This is a very effective attack and the attacker can implement it in a local network once he has access to it. The feasibility of a piece of malware performing this attack is lower. The setup of this attack is not trivial, as it is necessary to put the network card in promiscuous mode in order to sniff Ethernet traffic. In addition, the attack targets a specific local DNS, so it is not as easy as “start attacking anyone anywhere”. If the connection between a DNS server and the external servers is slow, the attacker will never be able to win the race and the attack will not be successful. Figure 3 shows the output of a working program in a Linux system performing the attack successfully.

```
francois@machine# ./sniffpoison -i eth0 -s 10.26.11.254 -d www.google.com -p 172.16.0.99

Listening on eth0 for DNS queries from 10.26.11.254...
Query packet found!
  Source IP:        10.26.11.254
  Destination IP:   192.58.128.30
  Source port:      53
  Destination port: 53
  Transaction ID:   0x424a
  Domain name:      www.google.com
Sending forged answer (172.16.0.99)...
Forged answer sent!

francois@machine# nslookup www.google.com 10.26.11.254
Non-authoritative answer:
Server: tipperary.ireland.tm
Address: 10.26.11.254

Name:    www.google.com
Address:  172.16.0.99
```

Fig. 3: Screenshot of test system sniffing DNS queries and sending the fake response. The target server (10.26.11.254) received a query to resolve www.google.com**. Our program sent a fake reply in order to poison the cache. A subsequent nslookup shows that the target server thinks that** www.google.com **points to 172.16.0.99, which is false. At this point, the target server will point any client to the wrong server as long as the cache is not flushed.**

Though it is a difficult kind of attack, once it is set up it is very reliable and that makes it a perfect attack for local networks. The only viable defense against this is to separate the corporate DNS servers in a DMZ, so that the attacker cannot sniff traffic between the server and the external DNS and thus the attack will never take place. It is worth mentioning that switched networks do not allow sniffing due to the very nature of that technology. In these kinds of networks, the attack is simply not feasible, as each dataflow travels separately and nobody can eavesdrop in the line. We suggest that network administrators implement these switching technologies, as not only do they allow higher bandwidth, but they also prevent any sniff-based attack.

DNS cache poisoning

This attack preys on the fact that non-authoritative DNS servers will always try to query the source of the real information. We call this a recursive query. In order to accelerate this information exchange, non-authoritative servers cache the information received so they can answer future identical queries without having to make the same recursive query again. This means that these ‘intermediate’ servers keep a local copy of the information and they do not usually delete it for some hours. Cache poisoning attacks focus on trying to change entries in the cached database. After this, the DNS server would reply to future queries with the wrong IP address. Figure 4 shows a diagram of the attack.

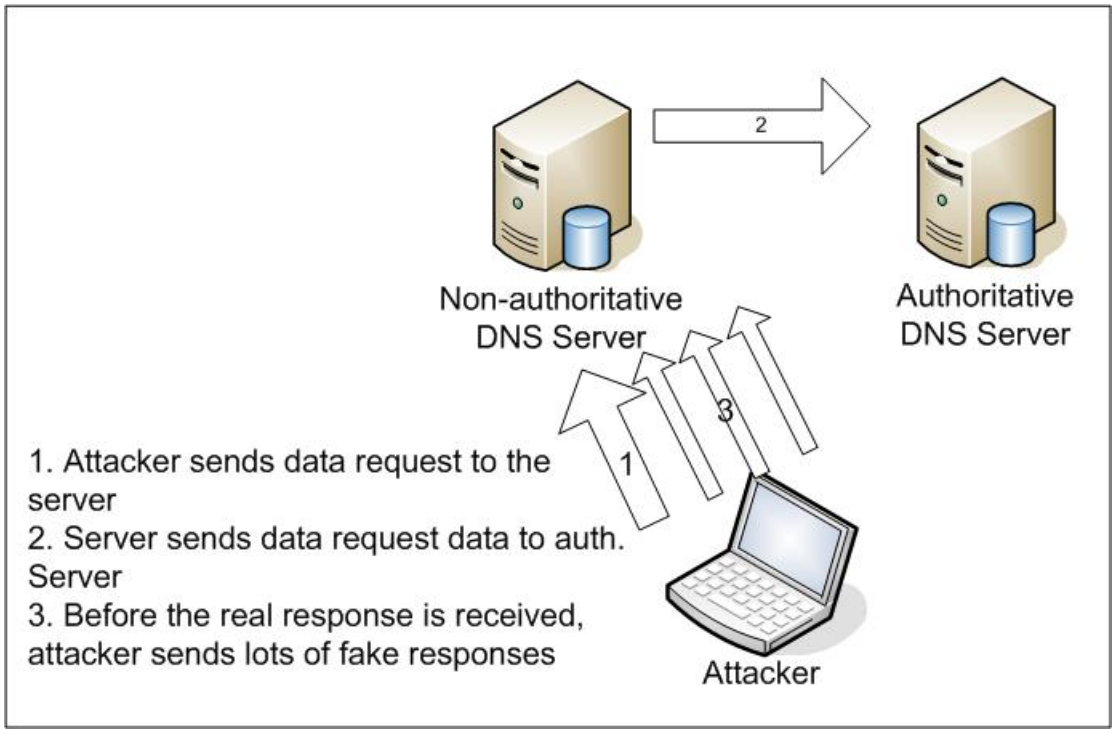


Fig. 4: Shows the flow of a Pharming attacker sending both the DNS request and the faked responses. If successful, the real response from the authoritative server will eventually arrive but it will be ignored.

Pharming attacks will always try to poison the local cache of non-authoritative servers so that future queries for the domain will point to a different IP address instead. During the time the cache is poisoned, the server will give the wrong piece of data to all users who contact it.

The technique to poison the cache of the server is based on a very easy concept. The attacker will send a simple request for a domain. Since the server is non-authoritative for that domain, it will try to query the info from the appropriate server. After that, the attacker will emit a fake reply as if it was coming from the real authoritative server. If the server accepts the fake reply as legitimate, it will cache the false information and will use the info to fulfill future requests. This will last until the administrator or the program itself flushes the cache, which can take some hours.

The creators of DNS already thought of this possibility. The DNS system has a specific protection to prevent the spoofing of responses. The way this works is as follows: there is a numeric field in the query called 'TransactionID' that can be set to any number from zero to 65535. The DNS server will need the response to have exactly the same TransactionID to prove that it is a real reply to a previous request. This is a good safety measure because it checks if the computer sending the response is the same one that our target server queried in the first place.

In order to perform the attack, the target server has to accept the fake response packet and that is by no means easy. So far, there are two documented instances when there is high probability of success for the attacker. These can happen thanks to certain software vulnerabilities in DNS servers, which make them create a predictable TransactionID. The key to the protection from receiving fake responses is this random number. Any flaw in its generation can ease the job of the attacker up to the point of allowing an easy prediction of the number. If this happens, the attacker can create the fake response that the vulnerable server will happily accept as true.

Birthday Attack Vulnerability

This server flaw refers to the server sending queries with a different random TransactionID for every identical request. Though this might seem to be a harmless thing to do, it creates a statistical effect known as the 'Birthday Paradox'.

The name stems from the counter-intuitive likelihood of two people in the same party having the same birthday. Though at first glance it would seem that the probability of this could be remote, it is actually quite high. This is because as soon as there are a few people in the party, the possible pairings grow exponentially. For example, if there are only two people, the probability is 1/365. If there are 10 people, the probability is 1/365 times all the possible couplings that can exist, which is 45. This yields a probability of 11.61%, which is still not high. If we try the same with 23 persons, the percentage is surprisingly high: 50.04%, this means that in a 23-person party, there is a 50% likelihood of two people having the same birthday. Not a very intuitive result, but true nonetheless.

The same mathematical experiment is true when we move the discussion to the Pharming attack. If the DNS server creates a new TransactionID for every identical request, the attacker can just start creating many requests and fake replies with random Ids. Like the members of the party before, the random couples will follow the same pattern. The number of packets needed to reach a 50% probability is around 300. This means that such an attack would be successful with a 97.6% probability by sending only 700 fake packets to the DNS server.

The formula to calculate the probability of these events is the following:

$$1-\left(1-\frac{1}{65536}\right)^{\frac{n\times(n-1)}{2}}$$

Fig. 5: The equation calculates the probability of a packet to find a twin one in a universe of N other packets provided that all of them are truly random. To use this in the birthday example, replace the range value 65,536 by the value 365. This is the number of possible different birth days.

Figure 6 displays the probability of success of this attack as compared with that of the brute-force method. Note how steep the birthday attack graph is and how it reaches almost a 100% success with a relatively low number of packets.

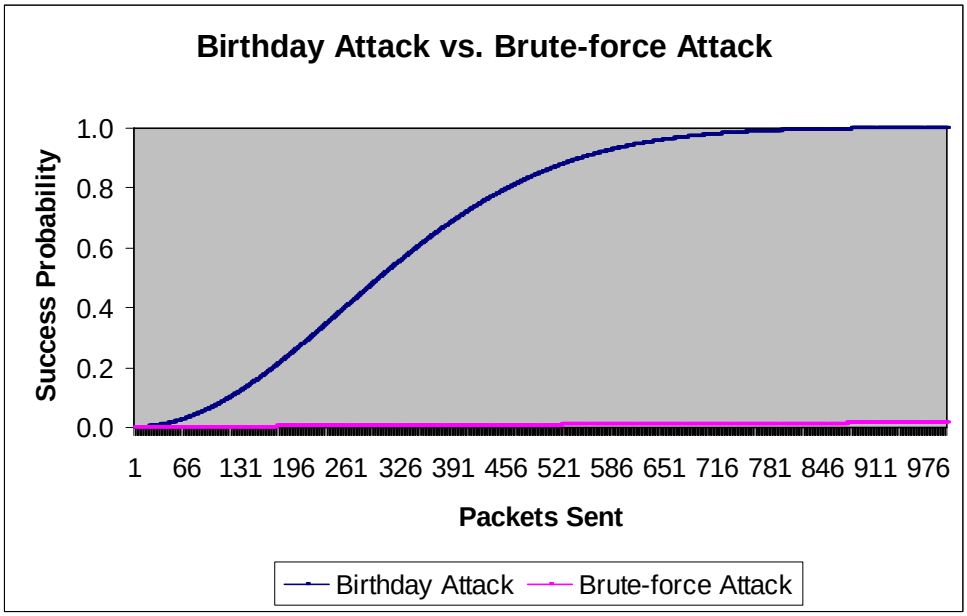


Fig. 6: Graphical representation of the probability of success obtained with the function of the Birthday Attack with a different number of packets used. As a reference, the graph below displayed the probability of the brute-force attack, whose probability would be N x 65,536.

All DNS vendors and developers have patched the affected servers in their latest versions by preventing them from using a different TransactionID for the same query. This effectively eliminates the possibility of using the Birthday Paradox to attack a DNS server for Pharming purposes. DNS server administrators must make sure that their servers are not vulnerable and they must keep them patched to their latest version at all times.

Random Number Generator flaws

Some time ago, researchers discovered certain flaws in the random number generator of some DNS servers. The challenge here is creating truly unpredictable random numbers. In computer systems, nothing is random and in order to generate different unpredictable numbers, programmers have tried many methods. They have tried using custom random sequences (pseudo-random number generators) and different random sequences every time (random number generators). The problem of using the first kind is that they become predictable once the attacker is able to extract the sequence. If the attacker queries these DNS servers repeatedly, he will eventually uncover the sequence. After analyzing the data, the attacker can predict the TransactionID with a 100% probability. Michael Zalewski wrote a research paper about this topic, analyzing the predictability of the randomness of different TCP/IP stack implementations and reaching some amazing conclusions. Taken from his study, the following graphs show three-dimensional displays of the results taken from the generators of well-known DNS servers in the market.

BIND 8

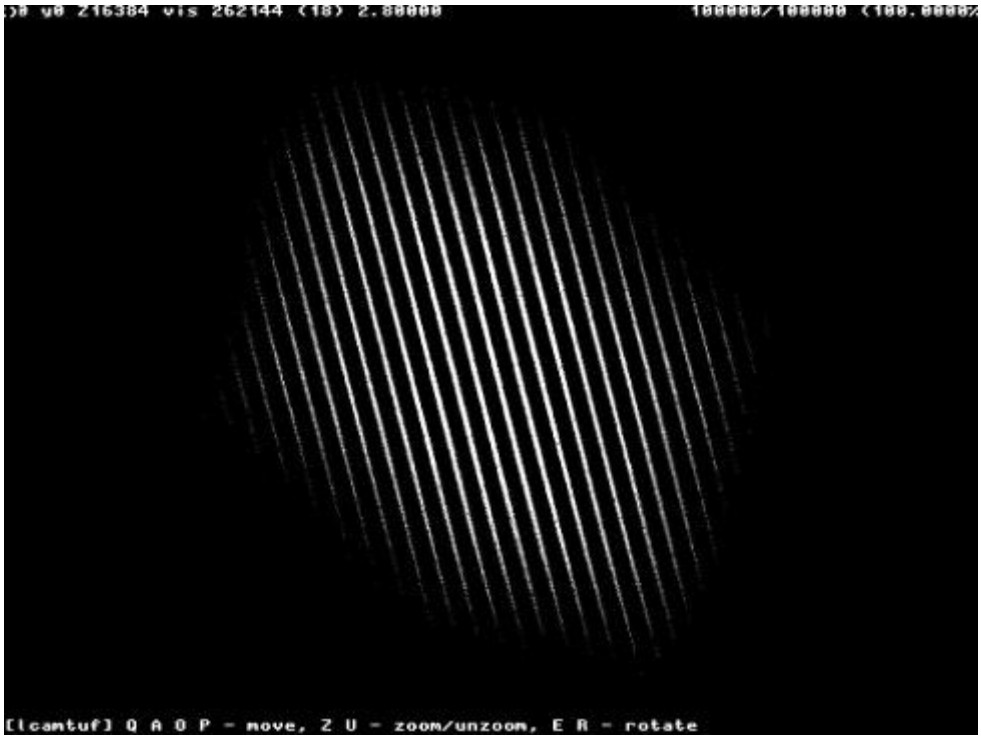


Fig. 7: 3D representation of the spread of the different TransactionIDs received from the BIND 8 DNS server implementation. It has been obtained from a test run of 100,000 responses.

A perfect random number generator would appear as an even cloud of points, perfectly spread out. In the BIND8 graph, there are strong attractors in the diagonal lines shown. Zalewski went as far as creating a program to predict future numbers. Based on the samples taken (and used to create the graph), the author could predict with a 100% likelihood the fourth number of any sequence. This means that after three failed attempts, the attacker could forge the server reply exactly and successfully poison the cache. This effectively threatens the implementation and makes it prone to the Pharming attack.

BIND 9

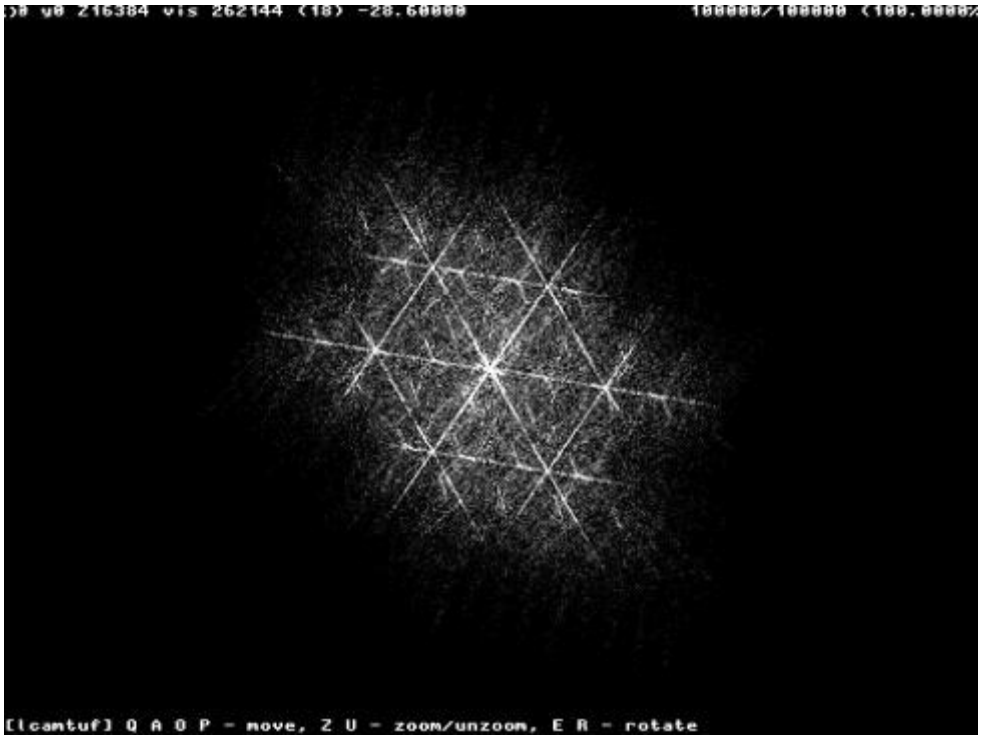


Fig. 8: 3D representation of the spread of the different TransactionIDs received from the BIND 9 DNS server implementation. It has been obtained from a test run of 100,000 responses.

BIND authors have used a new random number generator in their next version, based in the /dev/random device in the Linux operating system. Though this is a big advance, it is still predictable 20% of the time. This means that attacks directed against modern implementations of BIND 9 would succeed once every five attempts. This makes the attack not only plausible, but also likely if the attackers are members of a botnet. A distributed Pharming attack could be successful if properly targeted to the correct DNS server version. The difficulty of this kind of attack would be to ascertain the server version in order to send the correct targeted attack. This is by no means easy and it would work against the attacker, lowering their chances of success.

Djbdns

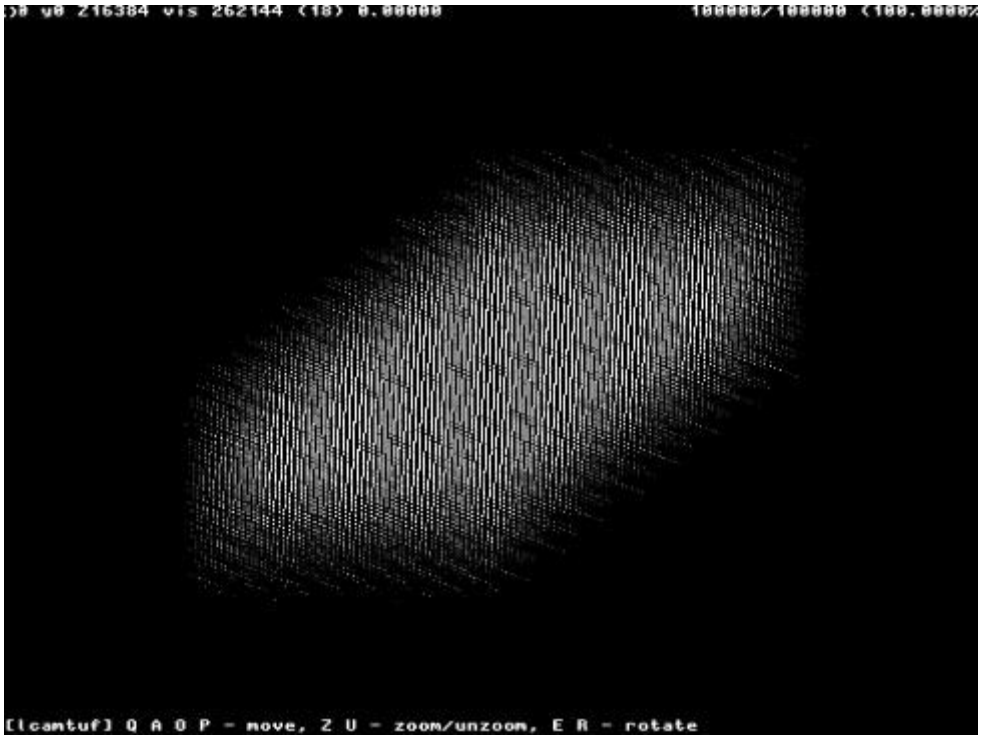


Fig. 9: 3D representation of the spread of the different TransactionIDs received from the DJBDNS DNS server implementation. It has been obtained from a test run of 100,000 responses.

Though this is a far less used DNS server, it is still popular in Linux environments. The author of the program, D.J. Bernstein is so confident about the security of the software that he has offered a monetary reward to the first person who reveals a security vulnerability in it. The graph shows a seemingly even cloud of points, but in fact, there are many attractors in the cloud, which make predictability a bit higher than in BIND 9. The probability of successfully predicting numbers in a given sequence is 30%.

If Pharmers targeted their attacks on the exact server software, Pharming would be a much more serious contender in the security arena. The existence of bias in random number generators proves that if the attacker knows the exact version of the target DNS server, he can perform a successful attack and redirect users to different web pages and to other services. There is only a theoretical proof of this. So far, Mr. Zalewski has created a program, which takes into account all the samples gathered and can predict the TransactionID with a certain probability. The mentioned program is not in the public domain; therefore, nobody has been able to use these concepts in an actual Pharming attack either in reality or in the lab.

Brute-Force attacking

As a third possibility, the Pharmer can also send all the 65,536 possible packets with the hope of guessing the correct TransactionID. This might seem to have a remote probability, but given the high speeds that Internet connections can reach these days, it is not so unlikely.

We have tested this kind of attack in our lab and we found that, while the average time taken to send all the packets is about 0.35 seconds, our window is somewhat lower, about 0.115 seconds. This window is the period of time when the attacker can send the packets: the remote query has

been sent but the real response has not been received yet. Though we took these data from a LAN environment, it is significant enough that we could fit an average of 32% of all the packets in the attack window. It is important to notice that an attack directed against an Internet host would be less successful. The reason is that with Internet speeds, we would be able to fit fewer packets in the attack window. In our experiment we also noticed how such an intense attack can leave the server in a very busy state. This means that while the attack is running, users would immediately notice packet drops and performance loss. This can lead to some sort of Denial of Service attack if the targeted server cannot process the onslaught of false responses fast enough.

Another interesting point we have to highlight from the test lab is that most of the DNS servers of well-known domains in the Internet are balanced. Usually, for each domain, there is more than one NS (Name Server) entry registered. For instance, a simple search for eicar.com yields two NS: *ns1.s-dns.de* and *ns2.s-dns.de*. An attacker faking the DNS reply could never know where it is supposed to be coming from. The attacker would need to choose one of those two servers to fake replies from. The chances of the attacker in this case would go down by 50%. Big companies have up to six DNS servers, dropping the likelihood to a mere 16%.

In our battery test, we chose over 100 different domains, which in total had an average number of 3.308 replicated DNS servers. This had to lower our chances by 30.22% in the average.

Out of 22,596 brute-force attempts, we successfully poisoned the cache 2,092 times. This is a success percentage of 9.25%. This roughly confirms our expectations of 32*0.303=9.7%

Our test still proves that brute-forcing is possible, though not likely. It also confirms that the only way this attack could succeed is with persistence: lots of packets, lots of times. This brings to mind the possibility of this attack being performed by zombies in a botnet. Nowadays bots are being used to perform Denial of Service attacks. It is easy to see that they could clearly be an effective platform for this kind of brute-force attack if they were properly modified.

Possible Solutions to the Pharming problem

DNS vendors are not only patching the known vulnerabilities, but also developing new systems to prevent this kind of attack. Microsoft, starting from Windows 2000 SP3 offers a new feature called “Cache Pollution Protection”, which specifically ignores caching for those responses where the NS record does not match the domain being queried. This is a step in the right direction and it can help prevent cache poisoning attacks.

From the user’s point of view, it is very difficult to verify if the server has been compromised or not. Attacks like “Cache poisoning” and “DNS spoofing with sniffing” are designed to be transparent to the user. The user cannot do much to prevent them and there is little chance he will even see them coming. As a development on the subject, Netcraft has developed an Internet Explorer bar with an interesting feature to prevent both Phishing and Pharming. It can verify and display the exact country where a server is situated. This can be of limited utility. If the user tries to access www.bankofamerica.com and the tool finds out that the server replying is located in Russia, it should be enough as to raise suspicion. This is an extreme example and in real life, servers used as Pharming ‘fake sites’ would be already compromised machines anywhere in the world. Since the attackers have no regard for region, the hacked machines can be anywhere in US or Europe. This would not be enough to raise suspicion to the bar user unless he memorizes where his usual servers are normally located. It can be of help, but it is not the ultimate remedy to the Pharming problem.

Fig. 10: Screenshot of the Netcraft plug-in in Internet Explorer. Note how it determines correctly the location of the server and provides some other data about the domain.

Conclusion

Overall, Pharming attacks do not seem likely to succeed. If DNS servers are properly patched, Pharming should be so rare as to raise little concern as a serious attack nowadays. The only other possibility that we have to rule out is the use of Pharming through Botnets. In the same way that zombies currently use Denial of Service attacks to flood servers, in the future they could try using Pharming in attempts to compromise DNS servers. The possibility of brute-forcing the attack does exist and if there is a tool that can outnumber the odds it is a large botnet. There have been instances of botnets containing up to 100,000 zombies. If the sheer force that such a bandwidth giant can gather was focused on attacking DNS servers, this could be the threat of tomorrow. In the meantime, all users can do is have an antivirus program running on their systems to prevent being infected by bots. They can also use the Netcraft bar to ease their hopelessness. The only definitive solution to nullify the threat would be to rethink the DNS protocol to make it use a longer TransactionID, moving from 16 bits (65,536 possibilities) to 32 bits (4,294,967,296 possibilities). This would completely rule out the brute-force method and eliminate the whole threat altogether provided that all systems were patched.

References

Zalewski, Michael (2001, April 21 and 2002, December 15). Strange Attractors and TCP/IP Sequence Number Analysis.

Joe Stewart (2003). DNS Cache Poisoning – the Next Generation. LURQ Threat Intelligence Group

Gunter Ollman (2005, July). The Pharming Guide. Understanding and Preventing DNS-related attacks by Phishers. Next Generation Security Software, Ltd.

Microsoft Knowledge Base (2005, April 12). Description of the DNS Server Secure Cache Against Pollution setting. Website: http://support.microsoft.com/kb/316786/

Cisco Catalyst 4500 Series Switch Cisco IOS Software Configuration Guide – Release 12.1.(13)EW (Chapter 19, pp. 19-1 to 19-5). Configuring DHCP Snooping

Netcraft Toolbar Website and program. Website: http://toolbar.netcraft.com/

The HoneyNet Project & Research Alliance (2005, March 13). Know your enemy: Tracking Botnets.

Spyware: A risk model for business

Jason Bruce
SophosLabs, Sophos PLC

About author(s)

Jason Bruce is the manager for detection development at SophosLabs UK. Jason joined Sophos as a virus researcher in May 2000 following a brief spell carrying out geophysical survey work for an oil field services firm.
Contact details Contact Details: c/o Sophos PLC, The Pentagon, Abingdon Science Park, Abingdon OX14 3YP, United Kingdom, phone +44-1235-544142, fax +44-1235-559935, e-mail jason.bruce@sophos.com

Keywords

Risk, model, factor, enterprise, potentially unwanted application, spyware, behaviour, functionality, consent, classifying

Spyware: A risk model for business

Abstract

The confusion of classifying adware, spyware and other labels for potentially unwanted technologies has lead to the creation of a risk model guide to help AV/AS vendors make decisions when determining an application's classification. This guide, the Anti-Spyware Coalition Risk Model Description, recommends the use of a risk model that weighs behavioural risk and consent factors to determine an overall classification and threat level for an application.

While this method of risk modelling may be appropriate for security products targeting the consumer market it is arguably an unnecessary complication for determining the types of application that are suitable for detecting in a business environment and could ultimately restrict business focussed security products from offering their customers the protection they demand.

This paper argues that for AV/AS vendors targeting business networks a simple assessment of the core functionality offered by an application is sufficient in the majority of cases to determine whether an application is suitable for detection and under what classification that application sits.

Introduction

The emergence of anti-spyware on to the security software arena has seen a change in the types of software being detected by the traditional anti-virus product. Gone are the days when an explicit malicious label could be attached to every detection. The spyware label it seems, can be attached to anything from an application deliberately intent on stealing a users private and confidential information, such as a banking Trojan, to neutral products, like port scanners, whose behaviour can only be judged based on the context in which they are used.

On the one side there is no change for AV vendors, they can go on attaching the malicious label to the likes of banking Trojans. However, moving through the spyware landscape we cross into a much hazier field of applications that are commonly labelled as potentially unwanted.

Due to the possible legitimate intent and sometimes commercial nature of these newly detected software types AV vendors have been required to provide customers with a choice as to whether certain applications are detected or not. Anti-virus vendors are not accustomed to providing this kind of choice to their customers. Equally customers are not accustomed to having that choice which means they need far more information and/or much clearer information to describe what it is that is being detected.

The challenge is to be consistent and clear about what is being detected so that customers and software vendors know where they stand. A risk model is a possible solution to this challenge.

A risk model for spyware

In the context of this discussion a risk model is a tool used to assess the risks attached to detecting and running an application in a given computing environment. The risk of detection is a risk carried by the AV vendor, the risk consequence being potential litigation by software vendors. The risk of running an application is carried by the user but is ultimately the responsibility of the AV vendor in the information they provide. The consequence of this risk is that the user is in some way negatively impacted by the functionality of an application that was not detected. To be successful a risk model must have enough factors to achieve its objective but not too many factors so as to create a confused noisy picture.

Ultimately a spyware risk model must be objective, clear and consistent. To achieve this though it may not necessarily be possible to make a risk model one size fits all and this paper will show that appreciable changes can be made to an existing model to make it more suitable for defining risks in an enterprise than on a consumer desktop.

Anti-Spyware Coalition

In January 2006 the Anti-Spyware Coalition (ASC) published a document called the "Anti-Spyware Coalition Risk Model Description". The purpose of the document was to describe a common process for risk modelling applications, as agreed by its members.

The model is based on the assessment of an application's risk and consent factors. The risk factors are the behavioural properties of an application that have the potential to be used in a malicious context. The consent factors describe how well a user is informed about the installation and subsequent usage of the application.

ASC documented risk-modelling process:

- Installation analysis phase - determine how the application is installed, looking for behaviour such as the level of consent that is sought.
- Application analysis – research the behaviour of the application and understand how it impacts the end user.
- Measure the risk factors – using the above analysis determine the risk level associated with the application.
- Measure the consent factors – using the above analysis determine the level of consent associated with the application.
- Weigh the risk and consent factors – consent factors can mitigate risk factors, this phase of the model takes this into account to determine an overall risk level. This phase also involves classifying the application type based on the risk factors brought out of the analysis.

The intention is for ASC members to agree on the overall process and the types of risk and consent factors that need to be considered. Members do not however necessarily agree on how those factors are rated and hence weighed against each other. This

immediately opens the model up to significant discrepancies between the products produced by anti-spyware vendor members of the coalition.

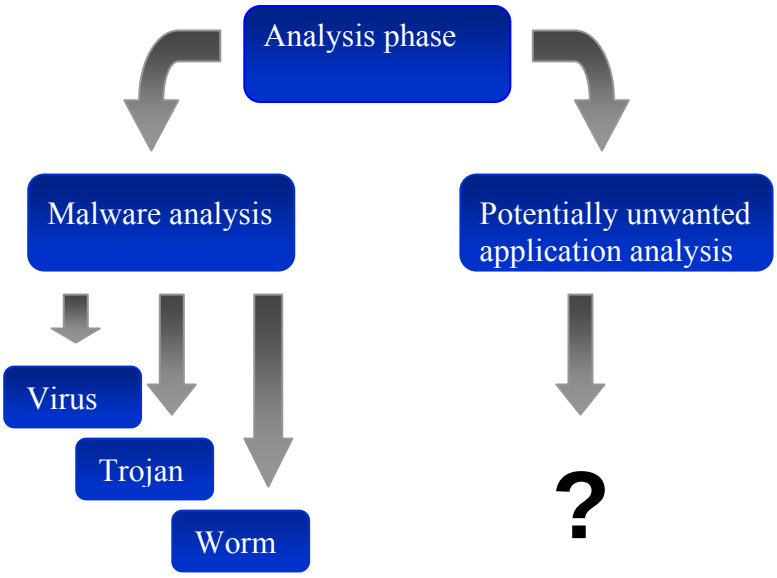
The risk modelling process described by the ASC is a good start at what is undoubtedly an important subject. However, with an ever-growing list of risk and consent factors to take on board it's possible that weighing these factors against each other could become an unwieldy task and ultimately unworkable on a day to day basis.

Potentially or definitely unwanted?

The ASC's risk model description attempts to encapsulate the full range of spyware definitions by accounting for malicious as well as non-malicious software.

Classifying and risk modelling malware

Anti-virus vendors have a great deal of experience identifying and analysing unwanted technologies, that is viruses, Trojans and worms. Viruses and worms for instance can be clearly defined by their ability to replicate. Trojans, while more difficult to define exactly, can be tied down by their behavioural properties, i.e. the context in which a function is delivered. Take for example the command "deltree c:\". Place this is a batch file and call it deltreec.bat and you have a legitimate tool, place the command in a batch file and call it tetris.bat and you have a Trojan that could delete everything in drive C:. When classifying an application as a Trojan, or more generally as definitely unwanted, the behaviour of the application is all-important.



Many AV vendors apply a threat level to all or some of the malicious threats they detect. Some take several factors into account while others take a more simplistic and arguably clearer approach of applying a single factor. The overriding factor governing the risk level of malware even where a vendor applies many factors is the prevalence of the malware in the wild.

There is no requirement for a spyware risk model to take on such a wide remit. Malware has been done and new models are not required to classify or risk assess malware any further. A risk model for spyware, whether targeted on the consumer or enterprise market should focus on the potentially unwanted element that exists in the various definitions of spyware.

Classifying and risk modelling potentially unwanted applications

When classifying an application that is potentially unwanted for the enterprise, the behaviour of the application is not as significant as the functionality that application delivers. Classifying an application as potentially unwanted is not to infer that the application is malicious only to say that a particular function of the application, clearly disclosed or otherwise may be unwanted on a particular computer or network of computers.

Let's say for the purposes of a very simple example that the functionality of deleting all files in drive C: is assigned a certain risk level. Using the simple Trojan example above, the application tetris.bat continues to be classified as a Trojan by virtue of its behaviour but the application deltreec.bat, whilst not malicious does present an element of risk by way of its functionality so could be classified as potentially unwanted.

As will be explained in the following discussion an enterprise focussed risk model for potentially unwanted applications can base its classification and risk assessment solely on functional characteristics, e.g. delivers popup advertising, runs as a remotely accessible server, scans a network range for vulnerable computers, etc. Appropriate levels of risk can be applied to each functional element that is explicitly called out in the model. The implementation of this functionality and the amount of consent and disclosure that is sought before running an application that delivers this functionality is not critical to a risk model targeted at the enterprise market.

Consent in the enterprise

What is significant about the enterprise and why do the requirements of a spyware risk model differ to those for a model representing the consumer market?

In the consumer market the user generally owns the desktop. The user makes all the decisions about the applications that are installed and run on the desktop and assumes full responsibility for how those applications are used. They can and perhaps should, from a competitive standpoint, be given the opportunity to make consented decisions regarding the applications that are allowed on their desktop without being unduly influenced by the external factors such as detection by an AV vendor.

In an enterprise environment any employee, whether they be a general user or an IT administrator, who installs new software and agrees to any accompanying licensing agreement is doing so on behalf of the whole company:

"… You are subject to the terms and conditions of this End User License Agreement whether you access or obtain the Product directly from the Licensor, or through any other source. For purposes hereof, "you" means the individual person installing or using the Product on his or her own behalf; or, if the Product is being downloaded or installed on behalf of an organization, such as an employer, "you" means the organization for which the Product is downloaded or installed and you represent that you have authorized the person accepting this agreement to do so on your behalf. …"

The ways in which employers manage the risk of employees installing software on their work desktops is quite diverse. They range from the full lockdown policy whereby users are given very limited access privileges and contractually prohibited from installing unauthorised software, to very loose policies where users have local administrator privileges and can install almost anything. The middle ground of an "acceptable use" policy appears to be the most common approach where users are permitted to install work related software with a written policy detailing prohibited actions.

Even the middle ground is a risky approach though as employers are entrusting users to make judgement decisions on areas that they do not necessarily have any expertise in. Leaving aside the risk of malware infections there are significant implications to the employer when potentially unwanted applications are installed, ranging from a drop in productivity to violation of laws enforcing the employer to provide sufficient protection of customer data.

In order to empower the IT administrator with better control of their network whilst not impinging on the access privileges that users have become accustomed to, the requirement to assess consent factors as part of an enterprise focussed risk model for potentially unwanted applications should be removed.

As an example, consider a user in an enterprise installing a remote access server application on their desktop. It is of little consideration to the administrator whether the installation and function of that application was disclosed to the user or not. Ultimately it presents a security risk to the enterprise by virtue of the functionality it provides and the administrator should always be aware of such an application being installed.

If a user were to request the installation of a new piece of software such as the one described above then there could be any one of three responses, depending on the policy for that organisation. An immediate yes or no may be the response based on the information provided by the user. Otherwise, as happens in some organisations, the application may be assessed and a decision subsequently based on that assessment. This practice is more common in companies such as IT security firms and large enterprises with dedicated security response teams (CERTs) where the expertise and technology for carrying out that assessment is in house.

What would be far more practical, secure and cost efficient to the enterprise is if they could trust their AV product to provide this information for them by way of alerting

on applications containing functionality that presents a potential security risk to the organisation. In the above example the administrator could make an assessment based on whether their AV product alerts on the application and if so what element of risk the accompanying description suggests the application may bring to the organisation.

Potentially unwanted in the enterprise

There are two main considerations to take into account when assessing whether an application is potentially unwanted in the enterprise.

Functionality that is potentially unwanted

This category includes functionality that generally has no place or purpose in an enterprise network despite being legitimate. It is not possible to say with absolute certainty that potentially unwanted functionality will not appear in any software that is used in the enterprise but it is possible to say with some certainty that the exceptions to the rule will be in the minority. Where a piece of software that is alerted on is required it can be authorised within the AV product.

Delivering pop-up advertisements is a classic example. Advertisements on employees desktops are of no value to the enterprise and offer only a distraction and unnecessary bandwidth load, both of which can impact on productivity. Saying that though it is still possible that an enterprise does make use of an application that contains this functionality in which case they can authorise that application to run.

Functionality that carries potential security risks

This category includes functionality included in software that may have a useful purpose in an enterprise but may also present a security risk if not used properly or used out of context, i.e. for a malicious purpose.

The remote administration tool previously referred to is a good example. Here we have a type of software that has a genuine legitimate purpose in many enterprises, however the potential to abuse or simply incorrectly set-up such software so as to present a security risk is high. Administrators should therefore always be aware of the presence of such software on a network.

Other examples of functionality presenting potential security risks are port and vulnerability scanning. Some network administrators use tools that include this functionality to check the integrity of their own networks. The existence of these tools on the network could however indicate the presence of a greater security threat such as a backdoor Trojan. A large proportion of security incidents originate from inside the company.

A risk model for the enterprise

There are a couple of key changes that would make the ASC's risk model description more appropriate for AV solutions focussed on the enterprise market.

- Remove the consent factors altogether as they introduce an unnecessary limitation on what can and cannot be detected. The consent considerations of the enterprise and consumer desktops are significantly different as has already been discussed. Consent in the enterprise is ultimately a decision that needs to be made by the enterprise's IT administration not the AV vendor or end user.

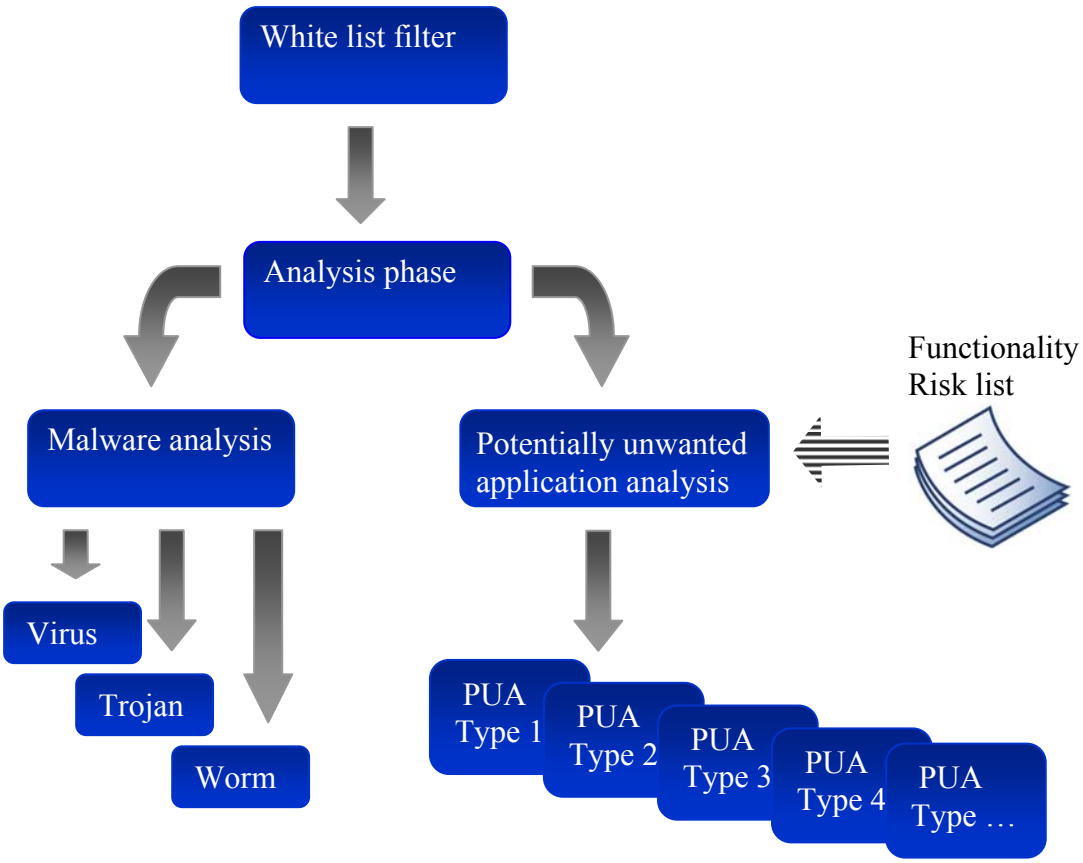
- Change behavioural risk factors to functional risk factors. Function can be discussed more objectively than behaviour, which as discussed is a combination of functionality and delivery. Certain functionality can present a risk to an enterprise regardless of how it is delivered if the appropriate authorisation has not been provided to use the application containing the offending functionality. Functionality is therefore the dominating factor of behaviour in the context of deciding what to detect in the enterprise.

A single "risk list" of functionality from the two main categories described above would be created for this risk model. A risk level using no more than three levels of risk to avoid unnecessary complexity would then be assigned to each piece of functionality listed. An application's overall risk score would correspond with the highest risk level associated with any of its functionality that matched those described in the risk list. The intention of this risk model is to alert the enterprise to software offering functionality that is either unsuitable in the enterprise or that brings an element of security risk. There is no intent to flag the software application itself as malicious or generally unwanted.

Just as in malware risk modelling where there is a single factor of prevalence dominating the overall risk level of an application so too with potentially unwanted applications there is a single factor that dominates the risk level, that factor being an application's functionality. Having a single factor greatly simplifies the risk model making it simpler for AV vendors to use and for customers to understand.

A useful addition to the modelling process would be the inclusion of a white list of software vendors who have particular prominence or high regard in the enterprise because of the enterprise level solutions they offer for their particular software area. This white list could be used to filter software applications in the first stage of the risk modelling process.

Potentially unwanted application risk-modelling process for the enterprise:



- White list filtering – Filter software based on the reputation of the vendor in the enterprise
- Analysis phase – Determine the broad classification of the application, malicious/non-malicious.
- Malware analysis – Unchanged from current practices. Determine malware type, behaviour and measure risk against prevalence data.
- Potentially unwanted application analysis – Determine the functionality of the application and compare this against a risk list of functionality that is generally unsuitable for the enterprise and/or introduces an element of risk into the enterprise.
- Classification and risk level – Using the above analysis and output from the risk list comparison classify the potentially unwanted application as a particular type (e.g. Adware, Dialer, Hacking Tool, System monitor, etc) and assign a risk level

Conclusion

The risk model described by the ASC attempts to account for all forms and definitions of spyware from malicious applications to those non-malicious applications that could otherwise be seen as unwanted. The model also attempts to take into account the full range of market focus by allowing AV vendors the flexibility to interpret the risk and consent factors in a way that suits their requirements. This attempt at a one size fits all approach to the risk model covers significantly more ground than is required by an enterprise focussed AV solution and because of this it is unnecessarily complicated for AV vendors to use and their customers to interpret. A clearer approach to the problem would be to have more than one risk model where each model has a particular market focus and fewer degrees of freedom so as to reduce the level of confusion.

This paper demonstrates how the risk model described by the ASC can be modified to create one suitable for an AV vendor focusing on the enterprise market. The scope of the original model can be reduced by taking advantage of the fact that AV vendors have many years of experience classifying malicious technologies. The complexity of the model can be reduced by removing the consent factors, one of the key factors of the original risk model but one which this paper argues is largely irrelevant when considering applications to bring to the attention of IT administrators in the enterprise. The complexity of the risk model can been reduced further still by considering the functionality of an application, when assessing the risk factors, as opposed to the, potentially subjective, behavioural characteristics considered by the original model.

There are limitations to this model, a significant one being the inability to account for actions not associated with the application itself but initiated by the software vendor prior to installation or download. The obvious one being vendors who make spurious claims such as informing users that their desktop is infected with spyware to trick them into purchasing their product. The risk-modelling process could be modified to accommodate this by reintroducing the specific installation phase analysis in the original modelling process described by the ASC. However it is questionable whether this is necessary for the enterprise risk model since these actions are not a risk to the enterprise and if the resulting product does introduce a risk then that will be picked up by the model as it stands.

The outcome of this discussion is a simpler model for enterprise focussed AV vendors to use since there is only a single factor to measure and one that is entirely objective. The model can also make a much clearer statement to customers as to why a particular application is being flagged as potentially unwanted.

References

Geer, D. (2004). Beware Spyware.
http://infosecuritymag.techtarget.com/ss/0,295796,sid6_iss486_art1014,00.html

Messmer, E. (2005). Spyware: Risky Business?
http://www.networkworld.com/weblogs/security/010248.html

Anti-Spyware Coalition (2006). Anti-Spyware Coalition Risk Model Description

Gordon, S. (2005). Exploring Spyware and Adware Risk Assessment. EICAR Proceedings 2005

Thompson, R. (2004). Minimizing Liability and Productivity Risks: How to Control the Impacts of Spyware, Hacker Tools and Other Harmful Applications. Computer Associates, White Paper

Whalley, I. (1999). Testing times for Trojans. Virus Bulletin conference proceedings 1999.

Spyware: Risks and Prevention

Martin Overton
IBM, UK

About Author

Martin Overton, IBM

Martin currently works for IBM [EMEA Managed Security Services Delivery (MSSD) core team] as a malware/anti-malware specialist, and is part of the Global Virus Emergency Team as well as the World-Wide Threat Team.

He is a regular speaker at the Virus Bulletin International Conferences, and has lost count of the many other presentations he has done and is a regular contributor to the Virus Bulletin periodical.

Martin is a charter member of AVIEN, a WildList reporter, a member of the Anti-Phishing Working Group and a founder member of the UK ISS User Group (UKISSUG).

To date he has accumulated over fifteen years of experience in investigating and combating viruses, Trojans and related malicious software (malware).

His hobbies, when time allows, include reading (mainly science fiction and science/technology/history books), astronomy, keeping a number of bugs (tarantulas and scorpions); and is a member of the British Tarantula Society. If this doesn't mark him as being weird enough, he also likes snakes (owning a Californian Kingsnake). Oh yes, and he does some computer programming. Occasionally his wife and son get to see him!

Contact Details: 51Cook Road. Horsham, West Sussex, RH12 5GJ, England, phone: +44 2392 563442, email: overtonm@uk.ibm.com.

Keywords

Spyware, Malware, IDS, IPS, Firewall, Policy, Education, Bots, Dialler, Trojan

Abstract

Spyware has grown over the last two years from a minor annoyance to what it is today; a major headache for companies and academia (most of them just don't know it yet) and home users alike.

This paper will investigate the growth of this threat and the 'cart-load' of risks and issues that Spyware and related risks bring to the corporate table. Furthermore it will investigate what the security staff in corporations can implement to address the risks and their company's liability, including.

- *Policy*
- *Education*
- *Firewalls*
- *Proxies*
- *Intrusion Detection Systems*
- *Anti-Virus tools*
- *And last but not least, Anti-Spyware tools.*

The processes, procedures and other solutions and guidance offered in this paper will come mainly from real-world experience of tackling spyware and related issues/risks.

Disclaimer:

Products or services mentioned in this paper are included for information only. Products and/or services listed, mentioned or referenced in any way do not constitute any form of recommendation or endorsement by IBM or the papers author.

Introduction

This paper will discuss spyware, what it is, some of the ways it gets on to your systems and finally what tools and methodologies you can use to combat it. Before we start let us cover a few definitions so that we all know what I mean by the relevant terms used in this paper.

I would strongly suggest that unless you have in-depth knowledge of Spyware and related malware that you try and obtain copies of the books/papers/articles listed in Appendix A.

What is Spyware

I will use the following definition:

"*Spyware is the generic name for any application that may track your online and/or offline PC activity and is capable of locally saving or transmitting those findings for third parties sometimes with, but more often without your knowledge or consent*".

If you want the full definition of what makes something spyware, then feel free to look here: http://www.antispywarecoalition.org/documents/definitions.htm However, don't expect it to be very concise!

Just like virus and other malware nomenclature, if you ask several experts, you'll probably get multiple and sometimes opposing definitions, you have been warned.

Spyware comes in many forms including adware, key loggers, trojans, browser hijackers, and diallers.

In fact most malicious spyware such as Trojans, key loggers, password and information stealers, bots and remote access Trojans have been detected by anti-virus products for a number of years. Spyware is now used to group these threats into a 'genus', although personally I would group them all in the genus 'malware' and be done with it. All the categories and sub-categories being banded about only confuse those that are outside the industry, including potential customers.

According to Wikipedia[1] "*The first recorded use of the term spyware occurred on October 17, 1994 in a Usenet post that poked fun at Microsoft's business model.*"

They go on to state: "*However, in early 2000 the founder of Zone Labs, Gregor Freund, used the term in a press release for the ZoneAlarm Personal Firewall. Since then, computer-users have used the term in its current sense.*"

Wikipedia also state the following about the creation of the first so-called 'Anti-Spyware' tool:

"*In early 2000, Steve Gibson of Gibson Research realized that advertising software had been installed on his system, and he suspected that the software was stealing his personal information…As a result of his analysis in 2000, Gibson released the first anti-spyware program, OptOut*"

This paper will focus on malicious spyware, including: Trojans, key loggers and diallers as well as other classes of malware which include or can install spyware functionality such as many bots, remote access Trojans, worms either as core functions, add-on modules, helpers or other downloaded files or components.

[1] Source: http://en.wikipedia.org/wiki/Spyware

Discussion

This section of the paper will discuss, whether spyware is a problem, and if so why. The common ways systems get infected by spyware. What the risks and consequences are of your system being infected by spyware. Then I will cover some of the tricks used by spyware to allow it to hide from anti-malware tools and regenerate itself when removed. Finally I will cover how big a problem it already is and suggest how much bigger a problem it may become.

Then we will move on to how to tackle this scourge. The solutions suggested will range from simple policies and procedures through to education and onto tools; both software and hardware based ones. This will include some generic solutions to help minimise the threat from malware and hackers as well as spyware.

Is Spyware a Problem?

Well, according to a number of surveys it is a problem, a BIG problem. The trouble is that many of those infected may not even be aware of spyware and/or that their system is infected. Furthermore they may be blissfully unaware that their browsing habits, at the very least, or their financial data or every key press they make is actually being recorded, and being sent to the malware/spyware authors to misuse, as they see fit.

- More than 33 percent of system crashes reported to Microsoft were found to be due to spyware.[2]
- Nine out of Ten PCs connected to the Internet are infected with spyware.[3]
- A spy audit report published by Earthlink and Webroot in 2004 found an average of 26.5 spyware traces are present on a given PC. In a six-month period, two million scans found 55 million pieces of spyware.[4]
- 92% of corporate IT managers at companies with more than 100 employees claim they have a "major" spyware problem.[5]
- A very recent study by Webroot claimed that Spyware cost firms $62B in 2005[6]

How do I get infected?

The main target for spyware authors is the same one as for the authors of other classes of malware, this being Microsoft Windows. There have been some cases of spyware being created for other operating systems such as Linux and Mac, but these are a mere drop in the ocean.

There are many ways to get infected with spyware; however the most common ways are via web sites that use scripting, known vulnerabilities or by the author relying on social engineering to get you to install their spyware. Spyware can also get installed as part of a free tool or utility that you downloaded or have installed from some media[7]. Let us look at each of these in turn:

The most common way that spyware gets installed on a computer is by the computer user installing it.

---

[2] Source: Microsoft

[3] Source: National Cyber Security Alliance, June 2003

[4] Source: http://www.webroot.com/company/pressmedia/pressreleases/20040804-spywarereport/

[5] Source: Web@Work Study, March 2004

[6] Source: http://www.scmagazine.com/uk/news/article/540680/?n=uk

[7] Such as CD, DVD, USB Media drives, PDAs, Smart Phones and last but not least floppy disks.

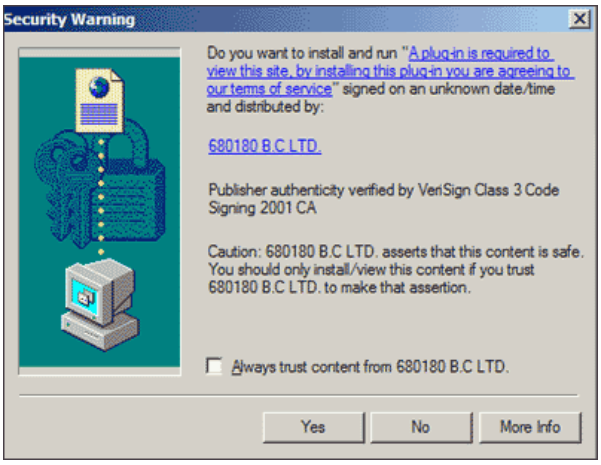
Web browsing:

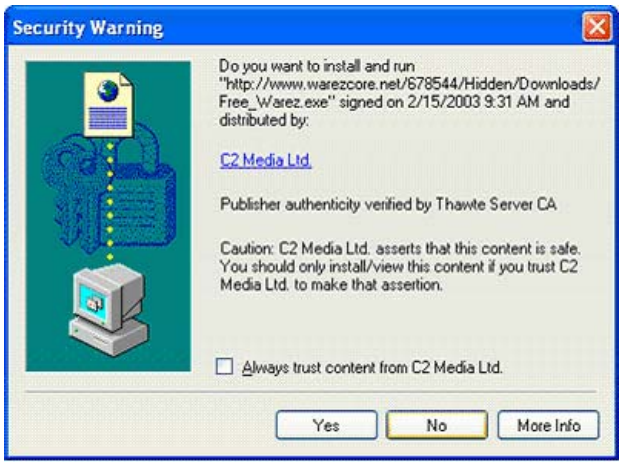
Spyware authors can also infect a system browsing a malicious website by exploiting vulnerabilities in the Web browser itself or the so-called helpers; plugins [usually ActiveX] such as viewers, java virtual machines, scripting languages such as VBscript or JavaScript [aka Jscript]. They may even take advantage of server side scripting languages such as Miva, PHP or CGI [such as PERL, Python, C/C++/C#, Unix Shell, Fortran, TCL and so on]. When the user navigates to a Web page controlled by the spyware author and/or cyber-criminals, the code on the malicious web page will attempt to attack the browser, or one of the helpers, and download and install spyware on to the visiting computer.

Why is this form of attack so common? Well, over the last few years we have seen the window between a vulnerability being announced and malware exploiting it shrink from years to months, weeks and more often now just a few days. So, this area needs to be addressed in the fight against malware and spyware as many use known vulnerabilities [which have patches available, and occasionally some that don't] to gain access to vulnerable systems.

Some of these vulnerabilities may be used when you visit a website which uses exploit code that your system is not yet patched against. These are commonly called 'drive-by-downloads' or 'drive-by-infections'. In some cases of these types of attacks, such as with the WMF vulnerability you may not even be aware that your computer has become infected. There is no warning, no download prompt, nothing to warn you or tip you off that something nasty and underhand has taken place during your visit to the site.

The next trick used by spyware authors is to use pop-ups to get you to install their wares, usually under the guise of some required plug-in or viewer, see *figure 1 and 2* for examples of what such a plug-in prompt may look like.





Figures 1 and 2: Plug-in Dialogue Box

As you can see this looks like just another dialogue box as you may be offered when visiting a web site that requires a plug-in installed for some of the content offered to be shown or for the page to work correctly. In reality these types of bogus plug-in download prompts when actioned, commonly install either diallers, key loggers or other Trojans.

Wikipedia have this definition of a dialler:

"A dialer (or dialler) is a computer program which creates a connection to the Internet or another computer network over the analog telephone or ISDN network. Many operating systems already contain such a program for connections through the Point-to-Point-Protocol (PPP).

Nowadays, the term "dialer" often refers specifically to dialers which connect without the user's full knowledge as to cost, with the creator of the dialer intending to commit fraud."

In the case of diallers they replace the original ISP dialler with their own, this then connects to the internet via a premium rate number using the telephone system via a narrowband modem, and this

can easily run up large bills. The first the user normally knows about it is several months later when their phone bill turns up.

The good news about diallers is that they can only work if you have a narrowband modem installed and you have the modem cable connected to a phone socket. This is even the case if you use broadband [DSL/aDSL/sDSL] normally. As long as the modem is plugged in it will force the computer to use the narrowband modem rather than the broadband connection.

This type of trick is not limited to those types of installations; they could just as easily be used to install a bot, downloader or mass-mailing worm.

The next two screenshots *figure 3 and 4* shows another common trick, the browser pop-up.

A pop-up is a type of window that appears on top of, and often over the current browser window of the web site that is being browsed. The other common type is the pop-under, which as the name suggests will appear behind the current browser window and is not often seen until the browser is minimised.

Clicking anywhere apart from the windows [X] control will usually either take you to the advertisers site, start a download or try and install a Browser Helper Object or ActiveX component which may be spyware or other malware. This is also commonly used to install bogus anti-spyware tools that are either spyware in their own right or will install spyware on to the system and detect it. The bogus anti-spyware will then inform the user that they have spyware on their computer and inform them that it can only be removed once they have paid for the anti-spyware program.

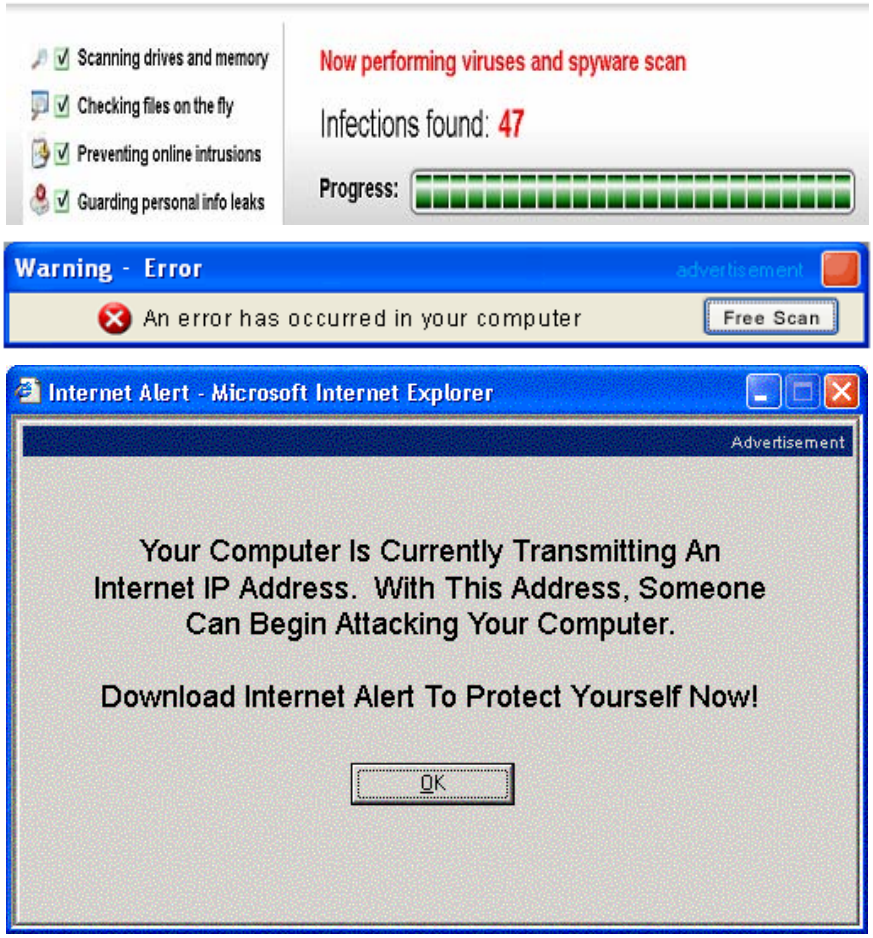


Figure 3, 4 and 5: Typical banner adverts and pop-up

Social Engineering:

Here's a short, but very apt, definition from the Jargon File: '*Social engineering n. Term used among crackers and samurai (hackers for hire) for techniques that rely on weaknesses in wetware (people) rather than hardware or software*[8].'

It is the human element that is the biggest risk, as no matter how strong your security, it is only as strong as its weakest link; the human behind the keyboard. "You [they] are the weakest link……in security" it is a shame we can't say "Good-bye!" to those that refuse to learn.

Furthermore, just to underline just how big a threat social engineering is, Bruce Schneier in his book "*Secrets and Lies*" lists social engineering as one of six 'aspects of the human problem' when focusing on information systems security. He states that social engineering is 'very effective', and that it goes straight to the 'weakest link in any security system: the poor human being trying to get his job done, and wanting to help out if he [or she] can.

In reality, however, social engineering is lots of things and it is even harder to pin-down when it is used in relation to malware, but the key to it all is the following: 'Someone wants something you have (or have access to) or wants you to perform an action (such as disclose information, run a program). To achieve this, the would-be Social Engineer will lie (claim to be someone or something they are not, or that they have access to something they are not entitled to), cheat (forge credentials or get you to run code that does something to escalate rights or install a backdoor by convincing you that it is something else) and steal (data, passwords, identities and/or availability of system resources)'.

In summary, the would-be Social Engineer plays on the natural human tendency to trust, and to want to help others, as a way to get you to do their dirty work for them.

Software:

Another common way for Spyware to get onto a computer is as part of an application which the user has chosen to download and install. This is a problem not just for freeware and shareware as is usually suggested; it can even come on some music CDs, just ask Sony[9]. To say that the Sony incident caused a media storm and a customer backlash would be an understatement[10].

Another area of concern is the growth of bogus anti-spyware software which is either spyware itself or downloads and installs spyware once it has been installed.

Those that use Peer to Peer [P2P] software are also at risk from spyware; just like with other classes of malware, bogus files are often found on these networks claiming to be useful tools, utilities, full applications or movies, music or pictures, which when run will infect the computer instead.

Vulnerabilities:

I have already covered vulnerabilities to a certain extent in the section on web browsing, however vulnerabilities are not limited to just web browser or associated helper application such as plug-ins, toolbars and so on.

Many bots rely on operating system vulnerabilities to force their way onto a new 'victim' computer. Some of these carry key logging or other spyware capabilities or can, once installed download them as requested by the botnet owner.

---

[8] Source: http://www.tuxedo.org/~esr/jargon/html/entry/social-engineering.html

[9] See here for one of the original reports disclosing that Sony was using 'rootkit-like' technology on a number of music CDs:- http://www.sysinternals.com/blog/2005/10/sony-rootkits-and-digital-rights.html

[10] See here for a good overall roundup of the issues, resulting media storm and finally the legal ramifications for Sony:- http://en.wikipedia.org/wiki/2005_Sony_CD_copy_protection_controversy

If you want more details on bots and botnets then I would suggest that you download and read a copy of my Virus Bulletin 2005 paper[11] on the subject.

E-mail:

It is becoming increasingly common for spyware to be installed by a downloader which is either spammed out or seeded by a botnet. This allows the spyware author to get a [usually] small executable out, which may be undetected by the anti-malware tools on many systems at that time.

Once the downloader Trojan has been launched it then often proceeds to disable or lower security settings in Internet Explorer, any anti-virus, personal firewall or other security software it can identify. Once the defences are reduced or neutralised the downloader Trojan will then invite in its friends. These could be bots, key loggers, browser hijackers and so on.

Below is an example of why and how this works using a castle as an analogy:

A soldier pretends to be a serf and gains employment in the kitchens of an enemy castle. Once in he is effectively invisible to the guards and immediately takes action. First he drugs the drink for all the soldiers and once they are all unconscious he acts.

He disables or sabotages all the defences he can, and then lets down the draw-bridge, signals to his comrades and their allies. They storm the castle and kill all the sleeping guards and their enemies who own the castle then proceed to rape and pillage at their leisure. The castle is now theirs.

Windows Messenger:

If your system has the Windows Messenger service running and it is not being protected by a personal firewall you may see strange messages. An example of such as message appears in *figure 6*, below.

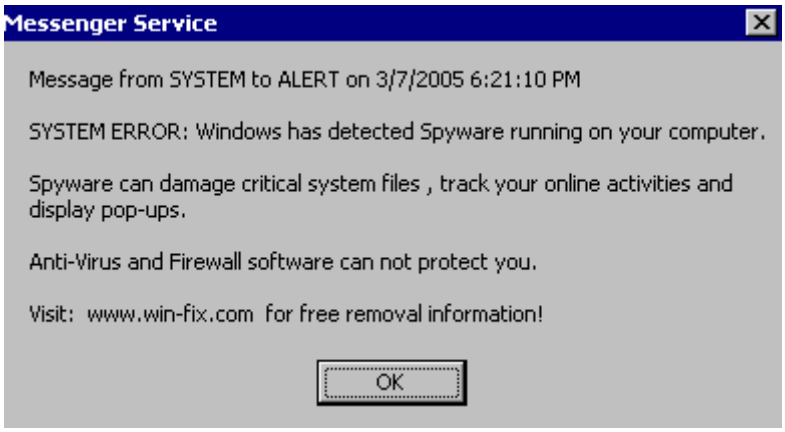


Figure 6: Typical Windows Messenger service pop-up

The Messenger Service was originally designed by Microsoft for use by system administrators to notify Windows users about their networks. However, some advertisers have started using this service to send information via the Internet.

All Microsoft Windows operating systems (98, ME, XP, 2000, NT) allow anyone on the internet to pop up Windows Messenger Service messages on your screen. The person sending the message may know nothing about your computer, where it is or what it is used for. It is just another target for their messenger service spam, or a way to get the recipient to panic and install their wares. Of course in many cases the wares offered contain spyware or other malware

Now let me make this perfectly clear, this is not, and has absolutely nothing whatsoever to do with the Microsoft instant messaging client, which used to be originally known as Windows Messenger and it now more commonly known as MSN Messenger.

[11] http://arachnid.homeip.net/papers/VB2005-Bots_and_Botnets-1.0.2.pdf

Other Malware:

Finally, as I've hinted at in the relevant sections above, once any malware has become installed on your computer it can easily invite in [download] and install, other malware including Spyware

So, to sum up, once spyware gets onto your system it won't be alone for long once one is installed it tends to invite [install] other spyware to keep it company, and to make some money on the side too for its trouble. In other words your computer becomes an open house to malware

Risks

Risk is a difficult area to cover. Most of us understand what constitutes a risk, however everyone's perspective on risk is different; one person's acceptable risk [such as mountaineering, motorcycle racing, bungee jumping, or keeping venomous animals (snakes, tarantulas and/or scorpions)] is another person's unacceptable risk. So, I will cover the areas of risk assuming that we are dealing with a company/institution/person that is inherently risk sensitive or averse.

What are the risks? Many are the same as other classes of malware and as such they fall neatly into the CIA triad:

CIA Triad Principle	Examples of how spyware affects the principle
Loss of integrity	System files modified, deleted or replaced Malicious code injected into system processes or files Security settings altered or removed Security tools modified, disabled or bypassed System used as a mule to store illegal and/or stolen material Data modified, stolen or deleted
Loss of availability	System running slower than before spyware was installed Unable to get to certain sites or re-directed to other sites Windows popup storms Reduced memory and/or disk space Reduced network bandwidth
Loss of confidentiality	Theft of credentials, such as passwords and other account details Intellectual property theft Financial data theft

What personal, financial and other data are the malware authors, and the cyber criminals that are often paying them to create the tools, currently interested in? The following for starters:

- Your name and address.
- Your telephone number(s)
- Your Social Security Number (or the equivalent used outside the US)
- Web site login data, such as Banks, Building Society, eBay, PayPal, ISP, etc.
- Credit-card numbers and .CVV or CVV2[12]
- Bank account details; account number and sort code
- CD Keys [Software Registration Keys/Numbers]
- Software registration keys

[12]More details on CVV and CVV2 can be found here:
http://support.worldpay.com/kb/product_guides/worldaccess/help/security_code_verification.html

- Passport details
- Driving license details
- Identity card details

So, really it is not so much Privacy at stake here, although that is a concern, what the malware authors and those creating and using spyware are really up to in this area is identity theft and fraud. In many ways this area is very similar to what the 419 [Advance Fee Fraud] and Phishing scams are trying to do. However, many bots and other spyware are using key logging techniques and/or are searching your hard drive for the data and handing it over without your knowledge, consent and without any assistance from you.

Identity Theft definition:

Identity theft results from the theft of key personal information. The perpetrator uses the information to create identification and credit cards. Identity theft often results in months of turmoil for the victim. Many people find that they need to restore credit ratings and to be freed from liability for illegal purchases.[13]

The following statistics about identity theft show a worrying trend[14]:

- According to 2 studies done in July 2003 (Gartner Research and Harris Interactive), approximately 7 million people became victims of identity theft in the prior 12 months. That equals 19,178 per day, 799 per hour, 13.3 per minute.
- The incidence of victimization increased 11-20% between 2001-2002 and 80% between 2002 -2003 (Harris Interactive). This same study found that 91% of respondents do not see an "end to the tunnel" and expect a heavy increase in victimization. 49% also stated that they do not feel they know how to adequately protect themselves from this crime.
- The Federal Trade Commission (FTC) reports that 27.3 million Americans were victimized by identity theft in the past five years, costing consumers $5 billion and businesses nearly $48 billion in 2002 alone.

So, as you can clearly see, the malware authors and those using spyware with key logging and data searching/stealing capabilities are in this for the money, not the fame or the intellectual challenge as has been used as justification for writing malicious code in the past. They have grown up from being the electronic equivalent of vandals and graffiti artists, and have become thieves, nothing more, nothing less. The really worrying part is that many malware authors are in the pay of organised crime syndicates and it can only be a matter of time before we see our first millionaire malware author.

Intellectual Capital

In many ways this is very similar to the issues with privacy and identity theft; except the data is not of a personal nature, it is sensitive or valuable information that can be sold to information brokers or may even be used to blackmail the company by threatening to send the information to a competitor unless they, the cyber-extortionists, are paid. They may even threaten to implicate a particular employee as being in league with them, either to get money or more data [intellectual property] from them or the company they work for.

Let us say that you work for a military, financial, medical or indeed any organisation or institution that has intellectual property that would be worth something to another party, let us say a competitor in your area of business or expertise. How much damage would it cause you personally or your company/institution if that data was copied, millions, billions?

---

[13] Source: http://www.cscic.state.ny.us/msisac/webcasts/05_05/info/glossary.htm

[14] Source: http://www.wholesecurity.com/threat/identity_theft.html

Many bots and other spyware classes have the ability to search for data. Some also open backdoors to allow full access to the file system of the infected system as well as any system that it is connected to, such as file and print servers. Imagine a firewall administrator whose system is infected; imagine if the firewall rule base was remotely modified by a malicious third party.

Spyware can be used to turn or microphones and web cams and record you or your meetings, scary huh?

So, what can I do to protect myself?

One thing you should be aware of is that there is no 100% solution to the spyware problem. Any company that informs you that their product offers 100% protection from spyware are either naive or just don't fully understand the real problem.

However if you approach the problem in the right way, then you can minimise the percentage gap from that perfect 100%. A well-designed approach can be expected to give a 98-99.5% protection from spyware and their effects.

The key thing to take onboard is that you or your company policy should advocate so-called 'Safe-Hex[15]'.

Policy and Procedures

Policies and procedures are the foundation of your company's security stance, it will also show how seriously you take security, or not as the case may be.

Just like foundations for a building, unless they are of a good quality and built on firm principles [or ground] then they will fail, crumble or sink without trace, leaving you or your company exposed to the elements; be they physical or technological.

The human element of security is the hardest to address, due to the general lack of interest in security which most end-users display, even to the extremes of openly flouting the rules and ignoring the security policies and procedures in place, much to the chagrin and disgust of the security staff that created them to protect their companies systems and networks[16].

Policies should not contain product names or detailed solutions, these should only be placed in procedure and technical solutions documents. All security policies and procedures [and related documents] should be reviewed at least once a year. This will enable new threats to be discussed and addressed and the relevant documentation updated to reflect this.

Your policy should include a statement informing staff that downloading and installing unauthorised software is not permitted. Any overrides of this policy should be signed off by the security department.

Use the K.I.S.S[17] approach for your anti-malware policy. The reason for keeping it simple is so that your staff can remember it.

Passwords

As many bots now have the ability to perform dictionary attacks to try and gain access to your system the need for good quality passwords, or even better pass-phrases is an absolute must. A dictionary attack is where a 'list' of passwords is tried, one after another until the list is completed, or access is gained to the system being attacked.

---

[15] You will find a good safe-hex description here:- http://www.claymania.com/safe-hex.html

[16] Source: You are the Weakest Link, Goodbye! – Malware Social Engineering Comes of Age – Martin Overton - Virus Bulletin March 2002 pp 14-17

[17] **K**eep **I**t **S**imple **S**tupid

I'm currently not aware of any bots or other malicious spyware which carry out brute forcing of passwords and/or pass-phrases; however I do expect this to happen before too much longer. A brute-force password attack would try every combination of numbers and letters [and maybe other non-standard ASCII characters too] until it gets access or runs out of combinations.

There are stand-alone tools out there that will perform these attacks, such as John the Ripper and Lophtcrack on both Windows and *NIX systems.

We may also see bots and other malicious spyware trying to steal password hashes instead of the plain-text password, as a number of tools exist that have pre-hashed [computed] thousand or millions of passwords/phrases and then it is just a matter of comparing the stolen hash against them until a match is found. These types of attacks are generally referred to as Rainbow tables[18]. As an example of the power and speed of this approach using a set of Rainbow Tables of 64GB which covers the following characters [`ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()-_+=~`[]{}|\:;"'<>,.?/`] and key space [7555858447479 possibilities which equates to 2^42.8] will break any 14 character password in a few minutes! [Based on at least a fast Pentium 4 based PC]

Now, imagine spyware which steals the LMHashes from Windows systems or Unix MD5 or SHA1 hashes and passes them off to such a system.

Education

I commented about the 'human problem' [aka Wetware] in a Virus Bulletin article back in 2002 which stated that "*the overall view of most end-users that security is an IT issue, and therefore not their problem. They seem to think that the technology will save them, what they really need to understand is that they are part of the problem, and are currently exacerbating it.*"

Education is important, but for most staff a simple security policy and acceptable use policy will be more effective than trying to educate them about all the types of risks out there on the internet. Instead focus on your support and technical staff, as they will probably be more interested and likely to retain the knowledge for a longer period. They may even end up by educating the end-users they visit and rub off some of their knowledge onto them; a bit like 'pollination' but without the mess.

Browsers

The first bit of advice I will offer is to use a browser that doesn't use/support ActiveX, as this is one of the main ways for spyware to get onto your system. I would suggest that you consider using Opera or Mozilla/Firefox instead. A recent study from the University of Washington found that "*Internet Explorer users can be as much as 21 times more likely to end up with a spyware-infected PC than people who go online with Mozilla's Firefox browser*".[19]

Don't get me wrong this won't stop all spyware getting onto your system via a web browser, but it should significantly reduce the risk of spyware getting on to your computer via a browser. Likewise, not visiting the internet's 'grey' areas or its seedy 'under-belly' will help, as many sites in these areas have bogus plug-ins and add-ons which are really spyware.

If you must use Internet Explorer then please make sure that you have it fully patched and set the security up correctly. At the very least set the Microsoft ActiveX support in IE to prompt you. To do this in Internet Explorer, click on Tools > Internet Options > Security > Custom Level, then click the check boxes that force ActiveX controls to ask permission before running.

---

[18] More details on Rainbow Tables can be found here: http://www.antsight.com/zsl/rainbowcrack/

[19] Source: http://news.yahoo.com/s/cmp/20060210/tc_cmp/179102616

Kill the Messenger

As mentioned earlier in the paper you should seriously consider disabling the Windows messenger service, as this will help to close one common avenue used to get your staff to install spyware and other malware onto their computers.

Instructions for disabling the Windows messenger service can be found here:

- Windows XP - http://www.microsoft.com/windowsxp/using/security/learnmore/stopspam.mspx
- Windows 2000 - http://www.microsoft.com/windows2000/techinfo/administration/communications/msgrspam.asp

Patch and Patch again

According to a new survey 'Two-thirds of U.K. businesses fail to patch[20]' their Windows desktops and servers. An older survey found 'Patch Management An Ongoing Challenge For Many Companies[21]' with 'only about one in five completely ready for the next virus attack'. Why is this a problem?

For home systems and those not already managed via third party or in-house patch management tools, you should at the very least ensure that all Windows systems are set to automatically check the Windows Update website at least once a week. If your systems run Windows 2000, 2003 or XP make sure you enable the Windows update service via Automatic Updates. This will ensure that updates are automatically downloaded and installed on those systems.

There have been a number of malware using so-called 'Zero-day' exploits. In this case there is no patch from the vendor to actually fix the hole in the operating system or application, and other mitigation techniques are required to partially or ideally completely manage the situation until a patch becomes available. An example of this would be the WMF exploit that surfaced in December 2005, but was not patched by Microsoft until January 2006.

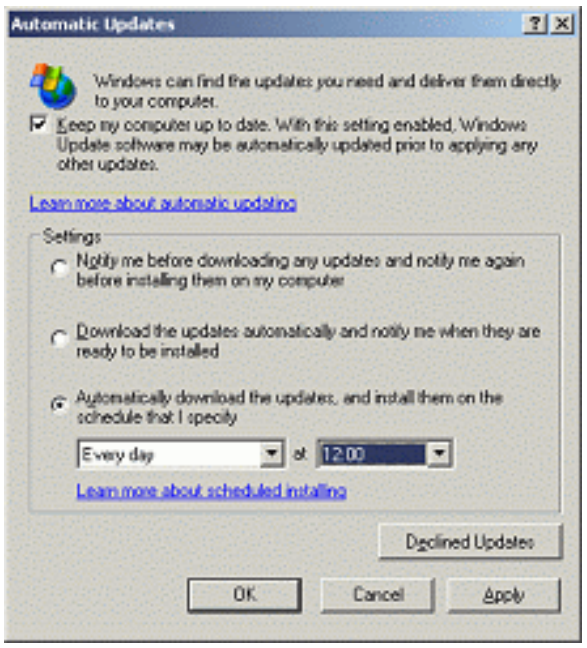


Figure 7: Windows Automatic Update Feature

[20] Source: http://www.scmagazine.com/uk/news/article/541973/?n=uk

[21] Source: http://www.informationweek.com/shared/printableArticle.jhtml?articleID=60405225

If you or your customers prefer to control when Windows updates are deployed across their networks then you could use the Microsoft Software Update Server [SUS] or its replacement WSUS[22].

Here is some data on WSUS from the Microsoft site:

Windows Server Update Services is a patch and update component of Windows Server and offers an effective and quick way to help keep systems up to date. Windows Server Update Services provides a management infrastructure consisting of the following:

- Microsoft Update: The Microsoft Web site that Windows Server Update Services components connect to for updates to Microsoft products.
- Windows Server Update Services server: The server component that is installed on a computer running a Windows 2000 Server with Service Pack 4 (SP4) or Windows Server 2003 operating system inside the corporate firewall. Windows Server Update Services server provides the features that administrators need to manage and distribute updates through a Web-based tool, which can be accessed from Internet Explorer on any Windows computer in the corporate network. In addition, a Windows Server Update Services server can be the update source for other Windows Server Update Services servers.
- Automatic Updates: The client computer component built into Windows 2000 with SP3, Windows XP, and Windows Server 2003 operating systems. Automatic Updates enables both server and client computers to receive updates from Microsoft Update or from a server running Windows Server Update Services.

There are lots of other third party patch management systems available, and some companies create their own instead of using off-the-shelf patch management tools.

Below are links to articles covering other solutions:

- http://www.networkworld.com/reviews/2003/0303patchrev.html
- http://www.serverwatch.com/tutorials/article.php/3414841
- http://www.serverwatch.com/tutorials/article.php/3381211
- http://www.serverwatch.com/tutorials/article.php/3424551

Perimeter and Network Firewalls

To help minimise the chances of infected systems ‘phoning-home’ once successfully infected by a malicious spyware you should ensure that you operate a ‘deny-all’ policy on your firewalls; both at the perimeter and also on other firewalls used to partition your network. If your company/institution does not allow IRC, which is widely used by bots, then ensure that this traffic cannot traverse your firewalls and network by using suitable filtering. For IRC start by ensuring that the default range of ports is not open, these being: 6600 – 7000/TCP.

The same goes for all other network aware applications that need [or want] to connect to the internet or across your own network, use a deny all firewall set-up. Only open up ports that need to be open for internet access. This will help not just in tackling spyware but malicious software in general which wants to connect to the internet to get new instructions, drop its key logs or download new versions of itself or other malware to replace it or supplement it.

Application Firewalls (Proxies)

Where possible proxy all traffic destined for the Internet, this includes IRC, HTTP, FTP and any other protocol or application that can be set-up to use a proxy server. All traffic for these protocols that do not use the proxies should be blocked.

[22] http://www.microsoft.com/windowsserversystem/updateservices/downloads/WSUS.mspx

Be aware that there are ways to make any application [even spyware] able to use a proxy, these include using Netcat, SocksCap, and HTTP-Tunnel.

If you do use a proxy, ensure that it is secured and you enable logging so that you can review the logs to look for any suspicious traffic which has passed through the proxy server.

Intrusion Detection Systems [IDS]

IDS Definition: "A system that tries to identify attempts to hack or break into a computer system or to misuse it. IDSs may monitor packets passing over the network, monitor system files, monitor log files, or set up deception systems that attempt to trap hackers[23]".

IDS comes in two main flavours; NIDS [Network based Intrusion Detection Systems] and HIDS [Host based Intrusion Detection Systems] and they both have a place in the fight against spyware especially bots and botnets. Then there is the offspring of IDS, known as IPS.

Back in 2003 the Gartner Group, caused something of a stir by pronouncing that Intrusion Detection Systems (IDS) and their Intrusion Prevention Systems (IPS) offspring were a market failure -- and in fact will be obsolete by the middle of the decade. I believe that the Gartner pronouncement was somewhat hasty in writing off IDS technologies.

The problem is not with IDS and IPS technologies; the problem is managing these tools and technologies and the massive amount of data they produce. Too many companies treated IDS and IPS as they did Anti-Virus; they installed them and left them to update themselves. Very few took the time to look at what the IDS/IPS was finding and fine-tuning them to get the best out of them or [even more worryingly] doing anything about the alerts that were being generated.

Host based Intrusion Detection Systems:

Most HIDS do one or more of the following to detect that a system may have been compromised:

1. Integrity checking
2. System Log monitoring
3. Policy driven behaviour blocking
4. Kernel wrapping
5. Buffer overflow detection

Network based Intrusion Detection Systems:

NIDS Definition[24]: Monitors all network traffic passing on the segment where the agent is installed, reacting to any anomaly or signature based activity. Basically this is a packet sniffer with attitude. They analyse every packet for suspected nefarious activity, most will also look for anomalies within the protocol.

There are many NIDS products on the market, probably the best known are:

- Snort
- RealSecure

SNORT can easily be augmented by either downloading extra signatures/rules or by creating your own. These signatures may also be able to be used [sparingly] with RealSecure's own SNORT signature support feature, known as TRONS.

A group known as 'Bleedingsnort' maintain numerous bleeding-edge signature/rule sets, not only for malware but new exploits and spyware. However the signatures/rules they produce are indeed 'bleeding-edge' and therefore may be more likely to cause false positives or even worse false-

[23] http://myphliputil.pearsoncmg.com/student/bp_hoffer_moderndbmgmt_6/glossary.html

[24] Source: http://www.networkintrusion.co.uk/ids.htm Also has other useful definitions, such as IPS, HIDS, etc.

negatives. Of course it all depends on the quality of the signatures/rules produced and the level of testing they go through to minimise both false-positives and false-negatives alike.

Bleedingsnort can be found here: http://www.bleedingsnort.com

My own Snort signatures can be found on my own personal website here: http://arachnid.homeip.net. However access is not available without requesting access to the relevant protected section of the site.

If you want to know more about creating your own signatures or installing running SNORT then I'd suggest you download a copy of my paper 'Anti-Malware Tools: Intrusion Detection Systems[25]' which was written for the EICAR 2005 conference.

Intrusion Prevention Systems [IPS]:

IPS Definition: An intrusion prevention system (a computer security term) is any device which exercises access control to protect computers from exploitation. "Intrusion prevention" technology is considered by some to be an extension of intrusion detection (IDS) technology, but it is actually another form of access control, like an application layer firewall.

Intrusion prevention systems were invented by vendors who decided to make access control decisions based on application content, rather than IP address or ports as traditional firewalls had done. This ability to inspect network traffic at a deeper level confused them with intrusion detection systems, which also inspect network traffic for signs of intrusions.[26]

Intrusion prevention systems may also act at the host level to deny potentially malicious activity.

According to some researchers, IDS is dead[27] and has been replaced by IPS. Examples of IPS products include: IntruShield from McAfee, Proventia from Internet Security Systems and Attack Mitigator from Top Layer. Just like with IDS there are both Network and Host based solutions available.

The beauty of IPS is that it can stop malicious traffic it recognises in its tracks, thereby stopping an infected system infecting others on the network. This includes spyware.

Enterprise Anti-Virus / Anti-Spyware

The use of anti-virus technologies as a detection method for systems infected by malicious spyware is self-evident, as many bots, key loggers, diallers are now reliably detected by anti-virus products.

Because of this we are seeing the inclusion of techniques in many of the modern bots and some other malicious spyware to allow them to disable as many security and anti-virus products as possible. In some cases this functionality may well be the first to be deployed, as a dropper being spammed out. Once run the dropper lowers or neutralises any local defences and then opens up the backdoor, or just downloads more components as required to complete the infiltration.

The thing to remember with anti-virus tools is that they can only [normally] detect malware they know about. New malware variants may well be detected by heuristics; however they are still far from perfect.

Many anti-virus vendors have bought in spyware detection technology, such as via an acquisition or licensing deals. Others have created their own and seamlessly integrated spyware detection into their existing anti-virus products. Either way it is good news for their customers.

---

[25] http://arachnid.homeip.net/papers/EICAR2005-IDS-Malware-v.1.0.2.pdf

[26] Source: http://en.wikipedia.org/wiki/Intrusion_prevention_system

[27] Source: http://www.esecurityplanet.com/views/article.php/2228631

A number of the major vendors in both the anti-virus and anti-spyware markets have had their products tested by independent third party testing bodies. One of these bodies is operated by Westcoast labs and is known as Checkmark.

Below you will find a table of products certified under the Anti-Spyware Checkmark.

Anti-Spyware Desktop[28]	Anti-Spyware Gateway[29]	Anti-Spyware Installed[30]
AhnLab Computer Associates Equiinet ESET Internet Security Systems Kaspersky Labs McAfee McAfee Consumer Panda Software International SOFTWIN SOPHOS Trend Micro Inc.	Aladdin Knowledge Systems Equiinet Finjan Trend Micro Inc.	Webroot Software Inc.

As you can see this includes not only desktop products but also gateway solutions. There are also hardware [appliance] solutions that can be used to combat malware at the perimeter of the network, these use a variety of techniques such as URL filtering, active content blocking or filtering many of these appliances are policy driven, so that you can decide what should and shouldn't be allowed in to your network. Examples of these devices include:

- Bluecoat WebFilter
- Finjan Vital Security™ Web Appliance
- McAfee Secure Web Gateway

A number of the largest anti-virus vendors offer products that can be centrally managed and will also offer compliance statistics for coverage and how up-to-date the signatures and products are within your network. Some of the management tools have been updated to manage spyware detection and personal firewall components alongside the traditional anti-virus functionality. This allows complete coverage of not only desktops but also servers and in some cases security appliances and other perimeter/network solutions.

The next section will look at some of the many anti-spyware tools that are available, and will really focus on home users solutions and tools that may be useful to support staff in organisations or academia.

[28] Source: http://www.westcoastlabs.org/cm-av-list.asp?Cat_ID=8

[29] Source: http://www.westcoastlabs.org/cm-av-list.asp?Cat_ID=11

[30] Source: http://www.westcoastlabs.org/cm-av-list.asp?Cat_ID=12

End User Anti-Spyware tools

If you want spyware protection for your home computer, bearing in mind that home users' computers are more likely to be infected than those in large businesses, then this is the section of the paper for you.

However, if you are looking for anti-spyware tools that might be suitable for use in a small to medium business or tools that may be useful for support staff; be they in small, medium or large businesses or even academia then this section should still be useful to you.

Microsoft may have come late to the anti-spyware party and by acquiring an anti-spyware company [Giant Anti-Spyware] and gate-crashed it, but they have arrived and everyone has noticed them, so lets get the introductions over with.

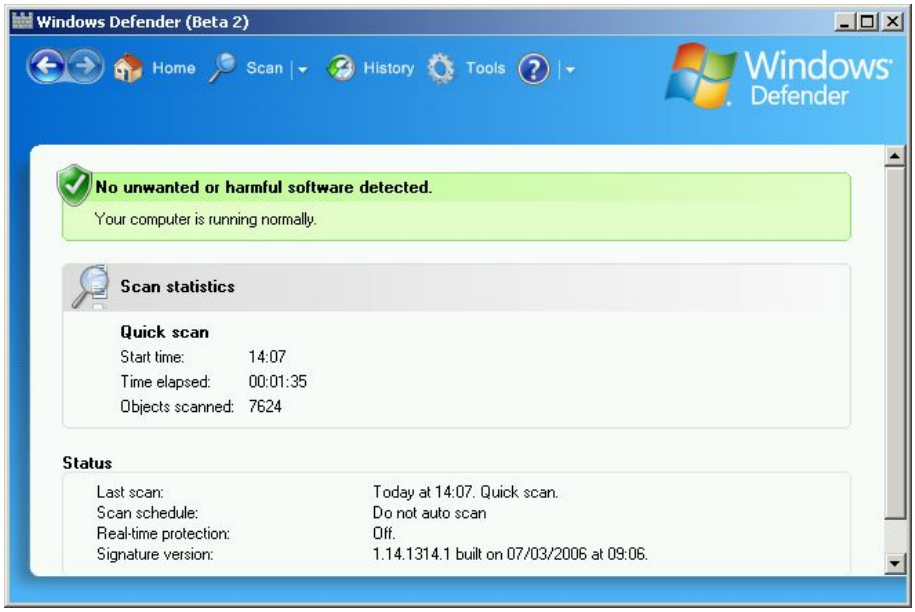


Figure 8: Microsoft Windows Defender (Beta 2) screenshot.

I have only tested it briefly and my results have been mixed, but is does seem to have reasonable levels of detection, however they do not, in my limited testing seem to be as good as either Adaware or Spybot Search & Destroy.

It does seem that Microsoft are taking spyware seriously, so given that this is a beta the finished product may well end up as a force to be reckoned with.

Microsoft has recently published a fact sheet[31] on Windows Defender.

Either way I would only suggest that Windows Defender is used alongside either Ad-Aware or Spybot Search & Destroy or an anti-virus product that has good spyware detection built in.

One of the anti-spyware tools I suggest that home users should consider is Ad-Aware. The product is easy to use, accurate and signature updates are regular. The free version will do on-demand scans and clean, however if you want on-access protection you will have to buy the Plus edition. This will get you the Ad-Watch on-access component that will block spyware as it tries to download or install.

[31] Which can be found here: http://download.microsoft.com/download/e/2/8/e28d34ec-4775-4626-ac19-a1b0e95e458c/DefenderBeta2FS.doc

Figure 9: Ad-Aware SE screenshot.

Likewise, I also suggest Spybot Search & Destroy to home users, and technical support staff too for cleaning up spyware infected/infested computers on their networks. Like Ad-Aware it works in two modes, on-demand and it also has an on-access component, known as Tea-Timer which not only will block spyware in real-time it also monitors the registry.

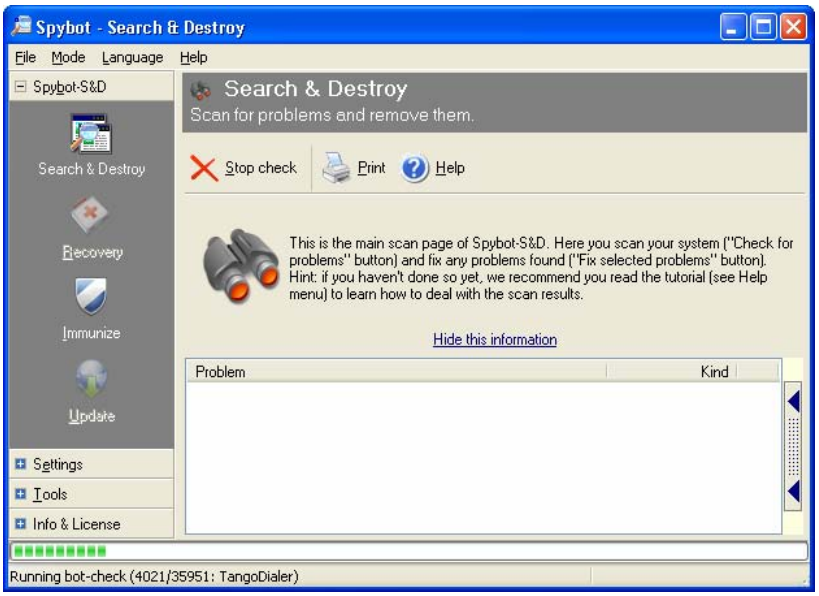


Figure 10: Spybot Search & Destroy screenshot.

Both of these anti-spyware tools well respected and updated regularly to detect new threats and are available in many different languages.

Before I finish this section of the paper, I would like to bring your attention to the fact that you need to be very careful when selecting an anti-spyware solution/tool, as there are a number of them that are spyware in their own right. You can find a list of the known 'bogus' anti-spyware and anti-malware tools here: http://www.spywarewarrior.com/rogue_anti-spyware.htm

Other useful tools:

This section is very much for the technical support staff that need to be able to diagnose and clean up the computers which get infected/infested with spyware in their organization.

Figure 11: WinPatrol screenshot.

WinPatrol is an interesting tool described by its author as a "*robust SECURITY MONITOR, WinPatrol will alert you to hijackings, malware attacks and critical changes made to your computer without your permission*."

It is a rather useful watchdog tool, as it monitors numerous parts of the operating system and key applications, such as Internet Explorer. It is also a useful diagnostic tool, not unlike HijackThis, which I will cover a little later on. However, unlike HijackThis, WinPatrol regularly checks the system areas monitored and warns you about any changes. You get to decide whether the change is allowed ot not.

It has functionality that is found in a number or individual diagnostic tools, such as Sysinternals autoruns[32] and a number of Windows tasks, such as displaying the current active tasks and services.

Another excellent diagnostic tool is HijackThis[33].

This is how the author describes it:

"*A general homepage hijackers detector and remover. Initially based on the article Hijacked!, but expanded with almost a dozen other checks against hijacker tricks. It is continually updated to detect and remove new hijacks. It does not target specific programs/URLs, just the methods used by hijackers to force you onto their sites. As a result, false positives are imminent and unless you are sure what you're doing, you should always consult with knowledgeable folks (e.g. the forums) before deleting anything*."

[32] Which can be downloaded from here: http://www.sysinternals.com/Utilities/Autoruns.html

[33] Which can be downloaded from here: http://www.spywareinfo.com/~merijn/downloads.html

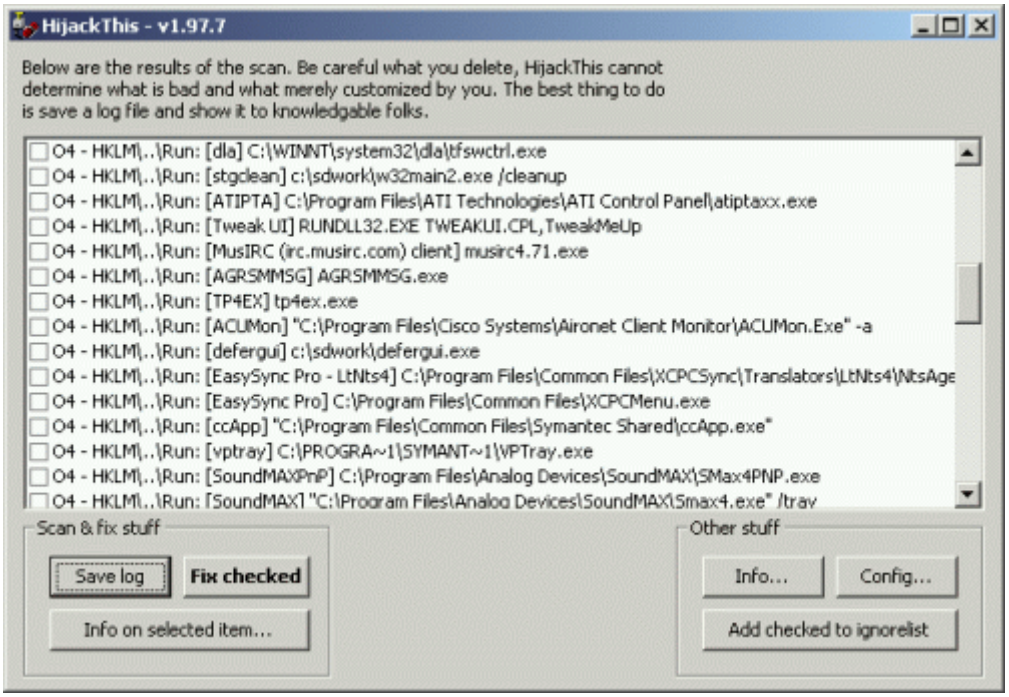


Figure 12: HijackThis screenshot

This tool is not for non-techies, luckily some kind soul has come to the rescue to assist in understanding the raw log files produced by HijackThis. This online tool is known as the 'HijackThis Log Analyser[34]'. This is a useful site for turning the output of HijackThis into something that means something to most end-users, not just techies or propeller-heads.

Occasionally no matter how good your anti-virus or anti-spyware tools are, you may come across some spyware that you just can't remove, or if you do, it just comes back again. The most pernicious of these is one known as CoolWebSearch. This spyware is constantly evolving and there seems to be an arms race in progress between the creators of the spyware and the creators of the anti-dote for it. This anti-dote is known as CWShredder[35].

CWShredder was acquired by Intermute which has now been acquired by TREND MICRO. It detects the new Cool Web Search variants, and CWShredder has been incorporated into the new Trend Micro Anti-Spyware 3.0

CoolWebSearch is the name given to a wide range of different browser hijackers. Though the code can be very different between variants, they are all used to redirect users to coolwebsearch.com and other sites affiliated with its operators.

If you want even more suggestions of how to make your system spyware proof or for more advice on how to tackle spyware, then I would suggest that you read this page: http://spywarewarrior.com/sww-help.htm

If you want to read just how sneaky some of the spyware can get, then I'd suggest you read Eric Chien's excellent Virus Bulletin 2005 paper entitled 'Techniques of Adware and Spyware[36]'.

[34] The HijackThis Log Analyser can be found here: http://www.hijackthis.de/en

[35] Which can be downloaded from here: http://www.intermute.com/spysubtract/cwshredder_download.html

[36] Which can be downloaded from here: http://www.symantec.com/avcenter/reference/techniques.of.adware.and.spyware.pdf

Anti-Rootkit Tools

Rootkits have been around for *NIX systems for many years; however they are now a growing problem for Windows systems. This is not only true in regard to bots and worms; we are now seeing Spyware authors actively using so-called 'rootkit' technology. This really should be called 'cloaking' or 'stealthing' techniques rather than 'rootkit technology' as what they are doing is hiding the malware files and processes from the operating system. Malware using stealth techniques is not a new phenomenon; many years ago DOS malware authors used similar techniques.

What is a rootkit?

A rootkit is a collection of tools an intruder brings along to a victim computer after gaining initial access, usually via hacking into the box manually or by getting the a user to execute a Trojan or Worm which will install a backdoor for them to slither onto the system in the first place. A rootkit generally contains network sniffers, log-cleaning scripts, and trojaned replacements of core system utilities. There is however another type which does not tend to replace system files, these are: Kernel [LKM] rootkits which subvert the system by attaching themselves to, or by otherwise modifying the kernel of the targeted operating system.

Some examples of such kernel rootkits on Linux include: Knark, Adore, and Rtkit.

Although *NIX rootkits have been around for many years and are generally considered the major threat to *NIX security, there are also a growing number of Windows rootkits. This 'rootkit' scenario is a complete about-face when compared to other classes of malware, where DOS/Windows is the most targeted and *NIX is little more than a drop in the malware ocean.

Some examples of Wintel rootkits include: Hacker Defender, FU and Vanquish.

Why do you need to consider rootkit detection tools? Well, a number of bots already include rootkit techniques[37] to allow them to hide from the OS and many security tools as they bind in directly to the kernel. A number of bots have used a recompiled version of the FU rootkit driver to remove their process entry from Windows Task Manager, others have used the JiurlPortHide driver for hiding network connections. It does look like we will be seeing increasingly sophisticated and 'invisible' bots and other spyware as rootkit technologies and techniques get added to the malware and spyware author's arsenal. A good example of this evolution is RIVARTS.A which is capable of running on Windows 98, ME, NT, XP and Server 2003 and uses rootkit [aka stealthing/cloaking] techniques to hide itself from the operating system.[38]

There are a number of tools available that claim to be able to detect and remove rootkits, these are listed below, along with the OS that they are suitable for:

- ChkRootkit [*NIX - http://chkrootkit.org/]
- Rootkit Hunter [*NIX - http://www.rootkit.nl/projects/rootkit_hunter.html]
- RootkitRevealer [Wintel - http://www.sysinternals.com/ntw2k/freeware/rootkitreveal.shtml]
- UnHackme [Wintel - http://greatis.com/unhackme/]
- Blacklight [Wintel - http://www.f-secure.com/blacklight/]

A number of anti-virus products now include so-called 'rootkit' detection functionality which is required to detect many of the more advanced ones that bind in at kernel level.

[37] More details can be found here: http://www.f-secure.com/weblog/archives/archive-052005.html#00000559

[38] Details can be found here : http://www.trendmicro.com/vinfo/grayware/ve_graywareDetails.asp?GNAME=TSPY_RIVARTS.A

Personal Firewalls

These can be used to block unwanted applications from being able to connect to the network, effectively, in the case of a bot, stopping it from connecting to the command and control network. In the case of spyware a personal firewall can be useful in stopping the spyware from connecting to the Internet and depositing its haul of stolen data.

For home users I would suggest either ZoneAlarm[39] or Kerio[40] as both are available as free versions, but if you want the extra protection or features then you can purchase a more advanced version of them.

Conclusions

Hopefully I have shown you if you have not already taken steps to counter spyware, then I strongly suggest you do before your computer [if you are a home user] or your network of computers [if you are a company or from academia] requires a complete rebuild to remove all the spyware and related cyber-threats that have infiltrated it and may well have stolen valuable personal, financial or intellectual property.

Spyware has been a problem for many years, and only the anti-spyware community, in the vast majority of cases were prepared to stand up and fight them on the battleground; this being your computer.

The bad news is that spyware is now commonly used by professional cyber-criminals to steal data, be it corporate secrets or your credit card or bank details. Even worse is that the quality of the spyware is getting better; this means that we are talking about these programs being written by professional programmers rather than the more usual stereo-typical malware author. Increasingly we are seeing new techniques to make the detection and removal of some spyware very, very, difficult.

The current trend in spyware to remain hidden and avoid detection means that up-to-date and multiple layers of protection have become almost mandatory for any self-respecting organization.

The good news is that not only do the dedicated Anti-Spyware tools and utilities do a good job in detecting and removing most spyware, with a few nasty exceptions, but the move from nearly all anti-virus vendors in including either better detection of spyware or offering anti-spyware add-ons is a very positive move.

As with other security threat, especially malware related ones, you need to deploy a multi-layered approach to minimise the chance of spyware getting onto you computers. This means not only do you need good technological solutions, and overlapping technologies at that, but these need to be backed up with good security policies, procedures, education and constant vigilance.

Please do not see this paper as an exhaustive or complete look at spyware, to do it real justice would require enough material to fill a large book.

---

[39] http://zonelabs.com

[40] http://www.sunbelt-software.com/Kerio.cfm

Appendix A – Suggested Reading

Implementing Anti-Virus [Malware] Controls in the Corporate Arena. Proceedings of the 16th Compsec International Conference, 1999 pp 575-586

You are the Weakest Link, Goodbye! – Malware Social Engineering Comes of Age, Virus Bulletin, March 2002 pp 14-17

Canning More Than SPAM with Bayesian Filtering, (Overton, Martin) - Virus Bulletin International Conference 2004

Anti-Malware Tools: Intrusion Detection Systems, (Overton, Martin) - EICAR International Conference 2005

Bots and botnets - risks, issues and prevention, (Overton, Martin) - Virus Bulletin International Conference 2005

Techniques of Adware and Spyware, (Chien, Eric) - Virus Bulletin International Conference 2005

The Common Malware Enumeration Initiative: An Update

Desiree Beck, Julie Connolly, & Michael Michnikov
The MITRE Corporation

About Author(s)

Desiree Beck is a Principal Information Security Engineer at The MITRE Corporation and the Common Malware Enumeration Technical Lead
Contact Details: The MITRE Corporation, 7515 Colshire Drive, McLean VA 22102-7508 USA phone +1.714.915.0310, fax +1.703.983.1002, e-mail dbeck@mitre.org

Julie Connolly is a Principal Information Security Engineer at The MITRE Corporation and the Common Malware Enumeration Program Manager
Contact Details: The MITRE Corporation, 202 Burlington Road, Bedford MA 01730-1420 USA phone +1.781.271.2325, fax +1.781.271.2633, e-mail jconnoll@mitre.org

Michael Michnikov is a Principal Information Security Engineer at The MITRE Corporation and a technical contributor to the CME initiative
Contact Details: The MITRE Corporation, 202 Burlington Road, Bedford MA 01730-1420 USA phone +1.781.271.2272, fax +1.781.271.2633, e-mail mbmg@mitre.org

Keywords

Malicious code, malware, CME, identifier, unique identification, antivirus, naming scheme, virus, worm, trojan, mass mailer, CME-24

The Common Malware Enumeration Initiative: An Update

Abstract

As is evident from numerous publications and an examination of online virus encyclopaedias, antivirus companies find it difficult to stay coordinated with virus names during major malware threat outbreaks, resulting in a variety of names and variant designations reported for the same threat. This results in considerable confusion, with members of the incident response community having to determine whether they are faced with a single threat or multiple threats, and most importantly, whether they are protected against the current threat(s).

Acting as a neutral third party, the United States Computer Emergency Readiness Team (US-CERT) is working in cooperation with public, private and international entities to adopt a neutral, shared identification method for malware threats. Known as the Common Malware Enumeration (CME) initiative, this effort seeks to reduce public confusion in referencing malware threats during outbreaks and to improve communication and information sharing between antivirus vendors and the rest of the information security community. CME is not an attempt to replace the vendor names currently used for viruses and other forms of malware, but instead aims to facilitate the adoption of a shared, neutral cross-referencing capability for malware.

This paper will provide a status report of CME since October 2005, when the initiative was publicly announced and the CME public website was launched (http://cme.mitre.org). Topics will include the role of the CME member groups (e.g., technical feedback group); plans for expanding the scope of the project beyond the current focus on high-profile malware threats; and a case study illustrating the value of CME to the security community.

Introduction

As is evident from numerous publications and an examination of online virus encyclopaedias, antivirus companies find it difficult to stay coordinated with virus names during major malware threat outbreaks, resulting in a variety of names and variant designations reported for the same threat (Brenner, 2004; Fitzgerald, 2002; Fitzgerald, 2003; Gordon, 2002; Gordon, 2003; Gordon, 2004; Mosby, 2004; Wells, 1999). This results in considerable confusion, with members of the incident response community having to determine whether they are facing a single threat or multiple threats, and most importantly, whether they are protected against the current threat(s).

Acting as a neutral third party, the United States Computer Emergency Readiness Team (US-CERT) is working in cooperation with public, private and international entities to adopt a neutral, shared identification method for malware. The MITRE Corporation manages the effort on behalf of the US-CERT. Known as the Common Malware Enumeration (CME) initiative, this effort seeks to reduce public confusion in referencing malware threats during outbreaks and to improve communication and information sharing between antivirus vendors and the rest of the information security community. CME is not an attempt to replace the vendor names currently used for viruses and other forms of malware, but instead aims to facilitate the adoption of a shared, neutral cross-referencing capability for malware.

Much has happened since publicly announcing the Common Malware Enumeration (CME) initiative in October 2005 (Brenner, 2005; Kuo, Beck, 2005; Shearman, 2005). In the past five months, CME has made considerable progress in achieving its initial goals and has begun to evolve the effort in some new directions. This paper will review the effort's current status, the obstacles that have been faced, lessons learned to date, and future challenges and directions.

Current Status

The original goals for CME were to:

- Assign unique CME IDs to high profile malware threats
- Create a community forum for sample exchange and deconfliction
- Put CME IDs in products and advisories

CME's initial operating capability assigns identifiers to high profile threats, currently defined as widespread malware threats and/or malware with significant press interest (Connolly, 2006; Kuo et al., 2005). To date, over 30 malware threats have been assigned CME identifiers. While this is a relatively small number, it correlates with initial expectations of approximately 1-4 high profile malware threats per month. CME focuses on high profile threats for three primary reasons. First, high profile threats generate the most public confusion. Second, restriction in scale may facilitate initial adoption. Finally, CME wanted to make an initial proof of concept capability available to the public while working collaboratively with the security community to ensure that the effort's scope evolved appropriately, matching the community's needs.

CME's initial operating capability also established a community forum for sample exchange and deconfliction. At present, that is carried out by the CME Sample Redistribution Group (SRG), a group currently comprised of more than eighteen organizations, primarily antivirus vendors as well as a few information security vendors and services organizations (Beck, 2006; Kuo et al., 2005). Members of the SRG identify and submit high profile malware samples to the CME submission server. CME relies on the historical experience of these organizations to know when an incident is considered high profile and noteworthy enough to be assigned a CME identifier. Once submitted, the sample is then redistributed to all SRG members along with an automatically-generated CME identifier[1]. With the focus on high profile malware threats, the need for deconfliction – or determining when two malware samples are considered the same – has been fairly low. Deconfliction has been the responsibility of the SRG members, who need to be aware of all previously-assigned samples when making a submission. Upon receipt of the sample and identifier, the SRG members are expected to start using the CME ID in their online malware encyclopaedias and other public communications. The CME public website publishes the ID, a short description of the malware, and a list of vendor aliases linked to their online encyclopaedia pages. This site also offers an RSS feed and an email list.

In addition to the participating SRG organizations, a number of other organizations have also started using CME IDs on their websites, in their advisories, and in other product offerings (Roberge, 2006). This is the ultimate goal of CME – for the identifiers to be used broadly – as it is the best way to help dispel confusion.

CME's first implementation phase has basically been completed: an initial operational capability for assigning unique identifiers to malware samples has been established; a public web site provides CME identifier and vendor alias information; and numerous antivirus and information security companies, as well as a growing number of user organizations, are actively participating in the

[1] If another sample has been submitted within the two hour window preceding the submission, an identifier is not automatically assigned. Instead, the later sample goes into "resolution" status; before an identifier is assigned, the submitter must verify uniqueness.

effort. Having achieved its initial goals, CME is poised to meet the next set of challenges as it continues to grow and evolve.

Challenges Faced

Although much progress has been made, the CME effort has faced several challenges. Perhaps one of the biggest challenges has been dispelling myths as to what CME is and is not. As the "virus naming problem" has been looming in this community for a number of years (Brenner, 2004; Fitzgerald, 2002; Fitzgerald, 2003; Gordon, 2002; Gordon, 2003; Gordon, 2004; Mosby, 2004; Wells, 1999), many assumed that CME was going to generate names that would replace the multitude of vendor-specific malware names. However, solving "the naming problem", or replacing the vendor-specific names, was never a goal of CME (Kuo, et al., 2005). Instead, CME has sought to provide a neutral point of reference to correlate all the vendor-specific names, similar to how the Common Vulnerabilities and Exposures (CVE) project has provided identifiers for vulnerabilities. Another misconception about the project is how, when and by whom the identifiers are assigned. Identifier assignment is triggered by an SRG member's sample submission. As noted earlier, CME relies upon SRG members' collective historical expertise to determine whether a particular malware threat meets CME's "high profile malware threat" criteria and warrants assignment of a CME identifier (Beck, 2006; Kuo et al., 2005). CME criteria for identifier assignment currently include widespread malware outbreaks and/or malware with significant media attention (Connolly, 2006). CME identifiers are assigned only when a threat is considered "high profile" per these criteria. Therefore, what may begin as a rather localized, mundane worm may not receive a CME identifier until it is touted in the press several days later. Educating the press and the community about CME has been and continues to be a major challenge.

A related challenge has been the relative speed at which the initiative has become known. While CME is far from having full acceptance, it has caught the attention of a significant number of individuals and organizations. Quick adoption was not MITRE's experience with the CVE initiative. CVE, as with most standards initiatives, took a few years to gain a solid foothold in the community, giving the CVE initiative time to mature and keep pace with expectations. By contrast, CME's fast adoption has resulted in some lofty and sometimes unmet expectations from the community. For instance, some assumed that CME was assigning identifiers to *all* malware and/or publishing significant analysis content. Certainly, growing the scope of CME identifier assignment and descriptive content is a goal of the CME effort, but the growth must be gradual and must be done using an iterative approach.

Two additional challenges that CME continues to face are (1) consistency in selecting threats for identifier assignment and (2) the changing nature of the threat environment. To date, CME has relied upon the historical expertise of its SRG members to identify those malware threats that are "high profile" and warrant a CME identifier so as to alleviate public confusion. This approach served CME well initially, but the growth of the SRG and the fact that samples are now being submitted by incident responders mean that a more formalized risk assessment approach is now needed. In addition, CME has had to recognize the realities of a changing threat environment. In the recent past, widespread pandemics were common, but today, malware threats are more likely to be localized, targeted attacks. Because CME was designed in the former environment, the effort will remain focused on high profile malware threats in the near term. However, in the longer term, changes must be made to ensure the initiative's sustainability, and such adaptations will likely prove an ongoing challenge for CME.

CME-24, A Case Study

In late January 2006, CME-24 was assigned to a mass mailing worm, reported to delete or alter files on the 3rd of each month. The assignment of the CME-24 identifier met the needs of the incident responder community by providing the unique, common, neutral identifier for the threat (see Figure 1), and it also raised a significant amount of awareness of the CME effort.

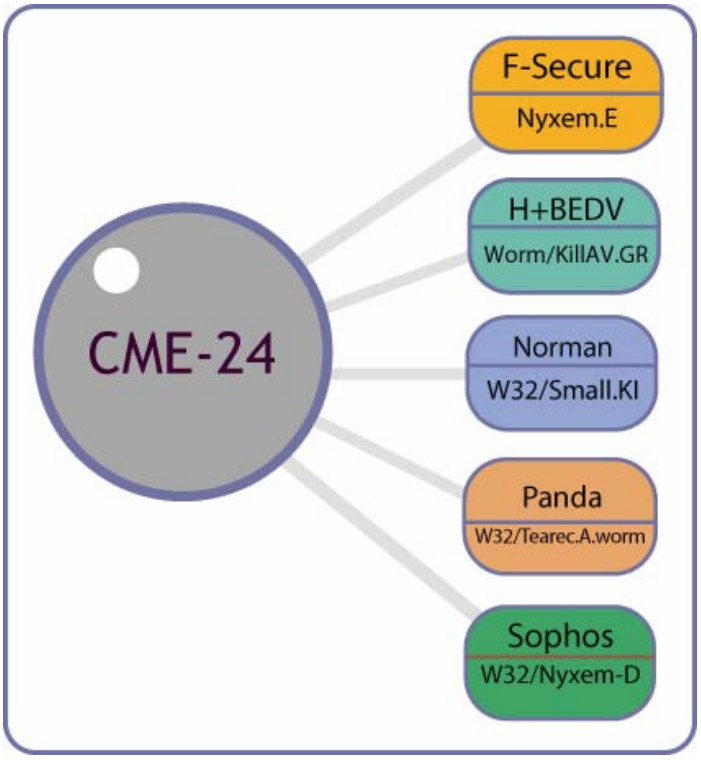


Figure 1. CME-24 Example

From the perspective of the CME organizers, the CME-24 incident was notable in two primary ways. First, the sample submission to CME came from the incident response community[2] instead of the SRG members. Second, several members of the incident response community referenced "CME-24" for the first time in their blogs, websites, and discussions with the press, and the majority of antivirus vendors added "CME-24" to their malware encyclopaedia entries (Evron, Stewart, et al. 2006; Foucart, 2006; Jesdanun, 2006; Krebs, 2006; Paller, 2006; Ullrich, 2006). As a result, many press reports used "CME-24" as one of their references to the worm (some reports even included a link to the CME website) and the CME initiative received a tremendous amount of publicity.

Despite what appeared to most participants as a successful execution, CME-24 did receive some criticism. Some thought that "CME-24" should have replaced the various names for the worm (e.g. "MyWife", "BlackMal", "Kama Sutra", "WORM_GREW"), instead of serving as a unique identifier to correlate these names (Krebs, 2006). A question also arose as to why the worm was first seen on January 17th or 18th, as reported by most antivirus vendors, but not assigned a CME identifier until January 24th (Krebs, 2006). Both of these questions pointed to the persistent confusion surrounding the current scope of CME and the identifier assignment process.

---

[2] A task force (Abrams, 2006) was formed to analyse the worm and notify affected and potentially affected organizations. This task force contacted CME, providing a sample and requesting a CME identifier.

It was emphasized that CME is *not* an attempt to replace the names currently used for viruses and other forms of malware, but instead aims to facilitate the adoption of a shared, neutral indexing capability for malware. It was also explained that CME-24 did not become a high profile threat until the task force convened and issued the press reports in the January 24th timeframe. While most AV vendors reported seeing the worm around January 18th, they reported a fast flurry of activity that trailed off quickly, and at that time, CME-24 did not appear to be a significant threat.

Future Plans

The primary goal of CME is to dispel public confusion during malware threat outbreaks. To fully achieve this goal, CME plans to grow the number and types of samples that receive CME identifiers over time. For example, in addition to assigning identifiers to viruses and worms, identifiers might be assigned to spyware as well. And, with the threat environment changing—with pandemics becoming less common and localized, targeted threats constituting the bulk of malware threats seen today—CME will likely have to deal with malware threats having thousands of variants. It is understood that CME will continually have to evolve in order to meet the challenges and expectations such as those that have been discussed previously. Progress will be gradual, with tangible progress being made through the accomplishment of a series of smaller, distinct, nearer term goals.

One central, near term goal of the initiative is to expand user community participation in the effort. Initially, CME focused exclusively on the vendor community, knowing that vendor use and propagation of CME identifiers was essential to the effort's viability. Once CME had established relationships with a number of vendor and related organizations, the CME Technical Feedback Group (TFG), comprised of leading computer incident responders and corporate IT security personnel, was established. While the TFG has started to provide valuable feedback, the participation of this user group is slated to grow. In particular, CME will seek TFG feedback on initiative directions and proposals; will ask that the TFG participate in the sample submission process; will welcome any analysis or other technical input that the TFG can provide; and will invite the TFG to actively participate in CME focus groups such as the "threat assessment" focus group discussed below. These are the users for whom CME was developed, and their direct involvement is vital if CME is to become a rich, valuable resource of the security community.

A second near term goal relates to international recognition and acceptance of CME. Recently, CME has requested that a Special Interest Group (SIG) be established under the Forum of Incident Responders and Security Teams (FIRST, www.first.org). FIRST represents computer security incident responders from government, commercial and educational organizations around the globe. In addition to broadening the awareness and use of CME by CME's primary target audience, FIRST's global affiliations should help extend international credibility to this US-sponsored effort.

In addition to the organizationally oriented goals involving the CME TFG and FIRST SIG, CME has technically oriented goals in the near term as well. One such goal is to develop a more formal and consistent approach for the identifier assignment process. As described previously, members of the SRG currently submit a sample for a CME identifier when they see fit, using their individual criteria for what constitutes a "high profile" threat. However, we hope to establish a more proactive approach, in partnership with the antivirus and incident response communities, to determine when a particular malware threat warrants a CME identifier. The plan is to further formalize the current CME identifier criteria of widespread outbreaks, damaging malware, and media attention, as well as identify threats independent of specific samples, obtaining the samples later if needed. This way, CME identifier assignment can be made in a more objective manner rather than falling to individual

SRG participants. This goal will be met through the establishment of a CME "threat assessment" focus group comprised of CME board, SRG, and TFG members.

Another goal is to publish additional descriptive information for each malware threat that is assigned a CME identifier. Currently, the content on the CME website consists of the identifier, the date it was assigned, a short description, and a list of vendor aliases with links to their online virus encyclopaedias. Additional information provided as an aid to incident responders might include (as applicable) affected ports and platforms, files, registry keys, CVE vulnerability information, and email subject lines. All information provided will be in a structured, XML format to enable ready consumption by automated tools. In addition, there are plans to make additional malware threat information available to a select group of users who would like to reference CME identifiers – such as on their sites or in their products – but who do not have access to the sample. Such information might include MD5 checksum(s), file size(s), and packer information.

Conclusion

The CME initiative has weathered its first five months in the public eye. It has caught a lot of attention, a fair amount of praise and some criticism. With patience, the growing participation of the user community, and the cooperation of industry and academia, CME will continue to evolve to achieve its tagline to "reduce public confusion during malware outbreaks."

References

Abrams, R. (2006). The Great Anti-Virus Conspiracy. Retrieved 1 March, 2006, from http://209.59.135.198/working_dev/threat-center/2006/02/great-anti-virus-conspiracy-working-in.html

Beck, D. (2006). The CME Sample Redistribution Group. Retrieved 1 March, 2006, from http://cme.mitre.org/community/srg.html

Brenner, B. (2004). Caught in the Virus Name Game. Retrieved 1 March, 2006, from http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1024919,00.html

Brenner, B. (2005). Will US-CERT Bring Sanity to Virus Naming? Retrieved 1 March, 2006, from http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci1130200,00.html

Connolly, J. (2006). Scope of CME Implementation. Retrieved on 1 March, 2006, from http://cme.mitre.org/index.html#scope

Evron, G., Stewart, J. and the rest of the TISF Black Worm task force (2006). CME-24 (BlackWorm) User's FAQ. Retrieved 1 February, 2006, from http://blogs.securiteam.com/index.php/archives/260

Fitzgerald, N. (2002). A Virus by Any Other Name: Towards the Revised CARO Naming Convention. 5th Association of Antivirus Researchers Asia (AVAR) Conference. Seoul, South Korea.

Fitzgerald, N. (2003). A Virus By Any Other Name: Virus Naming Revisited. Virus Bulletin. January 2003.

Foucart, S. (2006). Le virus CME-24 menace des centaines de milliers d'ordinateurs. Le Monde. February 2006.

Gordon, S. (2002). What is in a Name? Secure Computing. June 2002.

Gordon, S. (2003). That Which We Call a Rose.A. Virus Bulletin. March 2003.

Gordon, S. (2003). Virus and Vulnerability Classification Schemes: Standards and Integration. Retrieved 25 February, 2006, from http://www.symantec.com/avcenter/reference/virus.and.vulnerability.pdf

Jesdanun, A. (2006). Researchers Fear Confusion on Worm Name. Retrieved 3 February, 2006, from http://www.comcast.net/news/technology/index.jsp?cat=TECHNOLOGY&fn=/2006/02/03/318951.html

Krebs, B. (2006). Virus Naming Still a Mess. Washington Post. 4 February, 2006.

Kuo, J. and Beck, D. (2005). The Common Malware Enumeration (CME) Initiative. Virus Bulletin. September 2005.

Mosby, C. (2004). Open Letter to Anti-Virus Software Companies. Retrieved 1 March, 2006, from http://isc.sans.org/diary.php?storyid=356

Paller, A. (2006). SANS Newsbites. Volume 8 Issue 9. 31 January, 2006. Retrieved 1 March, 2006, from http://www.sans.org/newsletters/newsbites/newsbites.php?vol=8&issue=9

Roberge, R. (2006). Products and Services Including CME Identifiers. Retrieved 1 March, 2006, from http://cme.mitre.org/community/prod_serv.html

Shearman, J. (2005). Common Malware Enumeration Initiative Now Available. Retrieved 5 October, 2005, from http://www.mitre.org/news/releases/05/cme_10_05_2005.html

Ullrich, J. (2006). BlackWorm Summary. Retrieved 1 March, 2006, from http://isc.sans.org/blackworm

Wells, J. (1999). How Scientific Naming Works. Retrieved 1 March, 2006, from http://www.wildlist.org/naming.htm

The Trials and Tribulations of Testing Spyware Solutions: Towards a Testing Methodology

Larry Bridwell
ICSA Labs, USA

Abstract

ICSA Labs has been researching the problems of "Spyware" and the various technology and malware types that are referred to by that title. After many months of research and in-house testing we have learned that the technologies underlying "spyware" is basically neutral. Therefore, testing solutions for these forms of malware present many challenges. This presentation will discuss our path as we developed testing criteria, testing methods, and the review processes for determining test results. It will also present data (anonymized) from the testing of various solutions and the results of detection and "cleaning."

It is hoped that in presenting this material we can raise the level of discussion concerning spyware testing and reviews and move forward towards an acceptable industry standard testing methodology as we have in other technology arenas.

Unpacking: A Hybrid Approach

Vanja Svajcer & Samir Mody
SophosLabs, Sophos PLC

About Author(s)

Vanja Svajcer is a Principal Virus Researcher at SophosLabs, UK. Vanja joined Sophos as a virus analyst in 1998 after graduating from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. His interests include automated analysis, honeypots and research of malware for mobile devices.
Contact Details: c/o Sophos PLC, The Pentagon, Abingdon Science Park, Abingdon OX14 3YP, United Kingdom, phone +44-1235-540095, fax +44-1235-559935, e-mail vanja.svajcer@sophos.com

Samir Mody is a Senior Virus Researcher at SophosLabs, UK. Samir joined Sophos as a virus analyst in 2000 after graduating from St. John's College, University of Oxford with a Masters degree in Chemical Engineering, Economics and Management. His interests include automated analysis, generic and heuristic detection and research into packers and unpacking methods.
Contact Details: c/o Sophos PLC, The Pentagon, Abingdon Science Park, Abingdon OX14 3YP, United Kingdom, phone +44-1235-540230, fax +44-1235-559935, e-mail samir.mody@sophos.com

Keywords

Packer, Emulation, Unpacking, Scanning, Engine, Algorithm, Compression, Decompression, Encryption, Decryption, Plugin

Unpacking: A Hybrid Approach

Abstract

Malware obfuscation to evade detection has moved away from the use of complex polymorphic engines towards the simpler use of packing utilities (compressors, encryptors, protectors, self-extractors or combinations of the four). Usage of packing utilities can be a good indicator of malicious content.

Unfortunately, packing utilities are not malicious per se, so reporting a file as malicious on the packer code itself is not a good idea. There is a possibility, albeit small, that non-malicious software could also be packed with the same packer as used by malware. In addition, a variety of malware from different families can use the same packer. Misidentification is not an option. We must assume innocence until proven guilty.

The solution to the problem is to remove the layers of obfuscation to reveal the main file (the host) underneath. It would suffice to reconstruct a file that resembles the original in certain fundamental characteristics, eg the correct code/data and the correct PE entrypoint, ideally reconstructing the import table to help static analysis of the host's functionality.

Different approaches to unpacking have been attempted with the ultimate goal of achieving generic unpacking of the packed file. Emulation and tracing can be too inefficient and unreliable for inclusion in the AV product. Including code for new packers to the AV engine is efficient, but by the time the code is added and tested it may be too late.

Instead, we propose a hybrid solution based on reverse engineering the packer code to divulge the specific operations (functional blocks) that need to be executed to reveal the host, and then implementing the functional blocks in the detection data using a proprietary packer description language. This paper discusses the difficulties involved in and different approaches to unpacking with an example demonstration.

Introduction

Malware for 32-bit Windows platforms has existed for several years. During the initial period, malware samples were few and far between, consisting of simple overwriting, prepending or appending viruses. These early malware samples concentrated on malicious functionality rather than on code obfuscation techniques. They were easy to detect with signatures based on byte patterns, which is still the most common detection technique.

Of course, malware authors are well aware of the standard detection techniques. These malware authors began to make detection more difficult by writing more sophisticated code, both in the main functionality and in attempts to hide the malware's main code. Many malware authors began encrypting their malware files, some of these authors even resorting to complex polymorphism or metamorphism written in assembly. Detecting these files was no longer straightforward due to the lack of fixed bytes to include in signatures. Fortunately, incorporating effective polymorphic code generators into malware is not easy and the volume of different malware using these obfuscating techniques was relatively low. In addition, the total number of samples of malware was small by today's standards.

In the last two or three years the type of malware prevalent in the wild has changed markedly. Today malware consists mainly of mass-produced worms and Trojans written in a high-level language, rather than Windows PE infecting viruses written in assembly. Malware families have emerged where different members have very similar functionality, and usually have similar binary layouts. These samples can be detected with slightly more complicated signatures than before, where one carefully-written signature would detect multiple variants of the same family. Unfortunately, most malware samples attempt to evade detection by obfuscating their code. The obfuscation techniques are no longer based on complex polymorphic assembly written by the virus author. Instead, most malware samples use various freely available third-party packing utilities.

Packing utilities can be used to process an executable file, usually of type "Windows PE" (the "host") to produce another executable file, which has little, if any, resemblance to the original host. On the other hand, the packed files have essentially identical functionality when compared with the host. The packer is simply a wrapper around the host and there is eventually a transfer of control of execution to the host. Thus, malware files can maintain their functionality with obfuscated binary structures, which can make generic detection difficult since the host's code cannot be scanned without some processing to reverse the effects of the packer. To complicate matters, a single host file can have multiple layers of packing, each layer created by a different packer.

There is a wide selection of packers which have been seen being used to obfuscate malware, and the list appears to get bigger on a daily basis. Many of these packers are available for download via the internet, and their use spans various different, independent families of malware. Examples of these common packers include UPX, Petite, PECompact and ASPack, which have been seen being used to obfuscate everything from the Banker family of Trojans to mass-mailing worms such as members of the W32/Mytob family. Certain families of malware, like the W32/Bagle and W32/Sober families of worms, have resorted to using private custom-made packers. In this latter case the packer is unique to the malware, which is a rare scenario.

In most cases, packers have fixed bytes and can be detected by standard signatures. Unfortunately, however, a packer's code is not necessarily inherently malicious. Detecting on the packer's code alone would pose a serious false positive risk. Packers like UPX and ASPack are commonly used by clean files for the legitimate purpose of reducing the host size through compression. There are certain packers such as FSG and PEX which are rarely, if ever, used by completely legitimate applications, but we simply cannot judge a book by its cover if we want to maintain accurate, definitive detection of malware. For example, FSG has been used by inherently non-malicious software such as adware applications. In addition, there would be problems with nomenclature. Would every MEW packed file be detected as Troj/MEW? How would it be possible to differentiate between different types of malware (eg worm or Trojan)? In the case of malware families which use custom packers, detection can be reliably based on the packer code. However, custom packer code is usually modified greatly with every new release of a variant, so a signature for the custom packer merely provides protection from the variant which used it.

Packers come in a variety of flavours, usually involving compression, encryption or both. The encryption routines are generally unique to the packer and reasonably simple to decrypt. However, effective compression requires complex code, so compression routines in packers tend to be chosen from standard algorithms (eg JCLib, LZMA, APLib, ZLib etc) used in a multitude of common compression utilities.

Certain executable packers, such as LameCrypt, are straightforward as they encrypt the target host file in a simple way, but the vast majority contain relatively complex code (usually assembly based). In fact, several freely available packers are highly sophisticated in their ability to conceal the underlying file. Their combinations of compression and encryption functional blocks are preceded by anti-debugging and anti-emulation tricks and, potentially, polymorphic code. The encryption routines can be strong, industry standard algorithms like Blowfish (used in ASProtect), Rijndael (used in SVKProtect) or IDEA-based (used in Molebox). Even after getting past the millions of do-nothing polymorphic instructions and the anti-debugging code, breaking strong encryption is an extremely difficult and time-consuming task. Fortunately, in general, it is possible to get the source code for the strong encryption and decompression algorithms, which can be very? useful if implemented correctly.

There are a few well-known methods of extracting the underlying file from a packed one (the next section gives an overview of some of the common methods). Over the past few years there has been a lot of attention directed towards generic unpacking via emulation. Although emulation may work well for simple packers and, perhaps, even for some of the more complex packers, it is extremely unlikely to work well for the most complex of packers (Armadillo, Obsidium, ASProtect, SVKProtect, etc) which are used all too frequently for common bots like members of the Sdbot and Rbot families.

When considering alternatives to the use of emulation in unpacking, one important factor to consider is the bare minimum required from the unpacking routine. For example, the underlying host file returned after unpacking need not be in an executable state, complete with valid headers, imports, etc. Instead, it would suffice to return a file which simply contains the correct binary in common with the unprocessed original host, and perhaps the correct PE entrypoint, to provide a performance improvement for scanning purposes. Once the minimal portion of the host required is reconstructed, pattern matching with the existing standard signature is possible. If we analyse a packer's code, we could extract and implement its principal functional blocks. The common blocks would constitute an unpacking framework, driven by means of an unpacking language to extract the bare minimum required for detection. This process is time consuming and is not completely generic. Howevere, it certainly is effective for most packers. The successful integration of the framework with the AV engine provides distinct advantages over some of the traditional approaches to unpacking. A scriptable and portable framework allows for quick reaction to minor variations of the packer code, and gives performance advantages over some of the unpacking methods such as emulation.

Traditional Unpacking Methods

The unpacking problem has been historically tackled using techniques which can be classified as packer specific and generic methods.

Packer specific unpacking methods

Development of packer specific routines requires careful analysis of the packer code, to identify functional blocks and algorithms used for encryption and compression of the original host. The result of this increased analysis effort is a possibility for the packer specific unpacking routines to be optimised, to achieve better performance as they run native code in a native environment. However, packer specific routines suffer from susceptibility to minor packer changes, frequently

introduced by authors of malicious code to prevent the AV engine from removing the protection layer and revealing the real functionality of the host.

Static unpacking routines

Static unpacking routines are built-in as part of the anti-virus engine. Once the packer code is analysed, the unpacking code has to be implemented using a high-level language (usually ANSI C) to achieve portability of the engine over different processor platforms. Some anti-virus products support only Intel x86 architecture, and the developers have the option to improve the performance even further by hand-coding the unpacking routine in assembly.

Since the performance of the static routines can be optimised by hand, they achieve optimal levels of performance. However, because the packer recognizer code is compiled with the engine, sometimes even a minor variation of the code may cause the engine to misclassify files which the engine would otherwise be able to unpack without problems (e.g. modified UPX). The release cycle for the AV engine is usually a few orders of magnitude longer than the one for virus detection data, so once the engine misclassifies the packer it takes a fairly long time (sometimes months) before a new version of the engine which can cope with the modified packer is released. In the meantime, many other packer obfuscations may appear in the wild.

Another problem related to static unpacking routines is the memory consumption. Quite a lot of code may have to be duplicated, and with a large number of packers and their variants covered, the memory footprint may become a problem, especially in memory constrained environments like Windows CE.

Combination of static code and emulation

Static unpacking routines may be combined with the emulator to tackle more complex packers. Static routines have to be able to communicate with the emulator, extract parameters for the emulation, set registers and memory (with the potential to reduce the complexity of decryption loops) and instruct the emulator to emulate the required number of instructions. The emulator has to be carefully initialised to retain an acceptable level of performance.

Generic unpacking methods

In terms of unpacking functionality in the AV engine, a fully generic unpacker is the ultimate goal. Unfortunately, experience shows that unpacking is a task where accuracy and speed have to be balanced, as increasing one leads to the degradation of the other. Even if a generic unpacker able to unpack the most complex packers is built, the performance for some packers may still be unacceptable. Fortunately, the number of packed non-malicious files on an average system is small, so it is acceptable for the user to wait a bit longer while the product takes its time to process a packed file. The likelihood of a packed file containing malicious code is much higher than that for a non-packed file.

Emulation

Processor, memory and, to a certain extent, operating system emulators have been a part of anti-virus engines for a long time. Initially, they emulated 16-bit environments and were used to remove

layers of encryption used by DOS polymorphic viruses. They were also used to trace operating system calls to detect metamorphic viruses like ACG.

32-bit emulation was implemented soon after the first 32-bit polymorphic viruses, W95/Marburg and W95/Hanta, appeared at the beginning of 1998. Towards the end of the nineties most new malicious code was released as a monolithic executable, written in a high level language and compiled to native code. The implementation of a polymorphic engine using a high level language is still possible, but somewhat more difficult. The result was that only rare attempts at polymorphism in high level language malware were ever seen, and these usually ended up only in the collections of anti-virus companies as proof-of-concept samples. Virus authors were concentrating on replication over networks and the functionality, rather than the obfuscation, of code. At the time, replication over email was new and, in the absence of SMTP gateway anti-virus products, email was the most successful infection vector.

Some emulators evolved to the point of emulating the full personal computer, including the BIOS and ROM, inside the anti-virus engine, which allowed for dynamic analysis of a program based on the observed Win32 API and system calls and the side-effects of the infection process. In such an environment, the requirement to unpack files is somewhat less important, but the sandboxed environment introduces similar problems of suboptimal performance.

With the introduction of packers, emulation was the obvious method to employ to attempt the task of generic unpacking. However, several problems were encountered:

- Stopping problem – during the emulation of unpacking or any other executable code one of the main problems is to decide when to stop emulating and present the emulation buffer back to the AV engine. The end condition may be triggered by crossing the limit of emulation instructions acceptable for satisfactory performance. Other end conditions may include emulation of a cross-section (usually backward) jump, which is often seen once the host code is decompressed, or the emulation of host entry point code generated by engine-recognised compilers.

- Anti-emulation – packers often use anti-debugging techniques to prevent debugging, emulation and reverse engineering the host. Using OS structures such as PEB (process environment blocks) and mechanisms such as Windows Structured Exception Handling for anti-debugging purposes may also prevent the emulator from unpacking the file.

- Complexity of the processing algorithm – complex decryption and decompression algorithms require tens of millions of executed instructions. Since emulation is relatively slow, a large number of instructions makes emulation unacceptable in these cases. Typically, the slowest part of emulation is decoding an instruction, not the execution of the function that emulates the instruction. In the case of static loops, the performance of emulation can be improved by a factor of approximately 10 if the emulated instructions are cached. This is still not good enough for the most complex packers like Armadillo or SVKProtect.

Dynamic translation

Although a relatively well known technique, dynamic translation has not been used in many AV engines. The idea of dynamic translation is to decode packer instructions and translate them into a

safe format, with the possible removal of code constructs that would compromise the safety of the code if simply run natively on the processor. Once translated into a safe form, the code can be run natively on the processor, and thus achieve better performance than simple emulation. Since the code is usually run in a static loop, the translation has to be executed only once. Whilst the translation phase is somewhat slower than emulation, the speed improves considerably once the safe code is executed by native processors. Dynamic translation also helps with portability, since code that is unsafe for a source processor can be translated into safe code for the target processor (the one that actually runs the anti-virus program).

A Hybrid Method – Introducing DUL (Dedicated Unpacking Language)

Introduction

Unpacking a PE file does not have to be restricted to either emulation or static algorithms in an AV engine. Sometimes the desired effect can be achieved in a more efficient manner, by using a combination of different strategies to provide a flexible solution.

If a packer's code is reverse-engineered and split up into functional blocks, it is possible to simulate the packer's code in an efficient and convenient way, using functionality in an AV engine, to create an unpacked "host" with appropriate physical characteristics for scanning purposes.

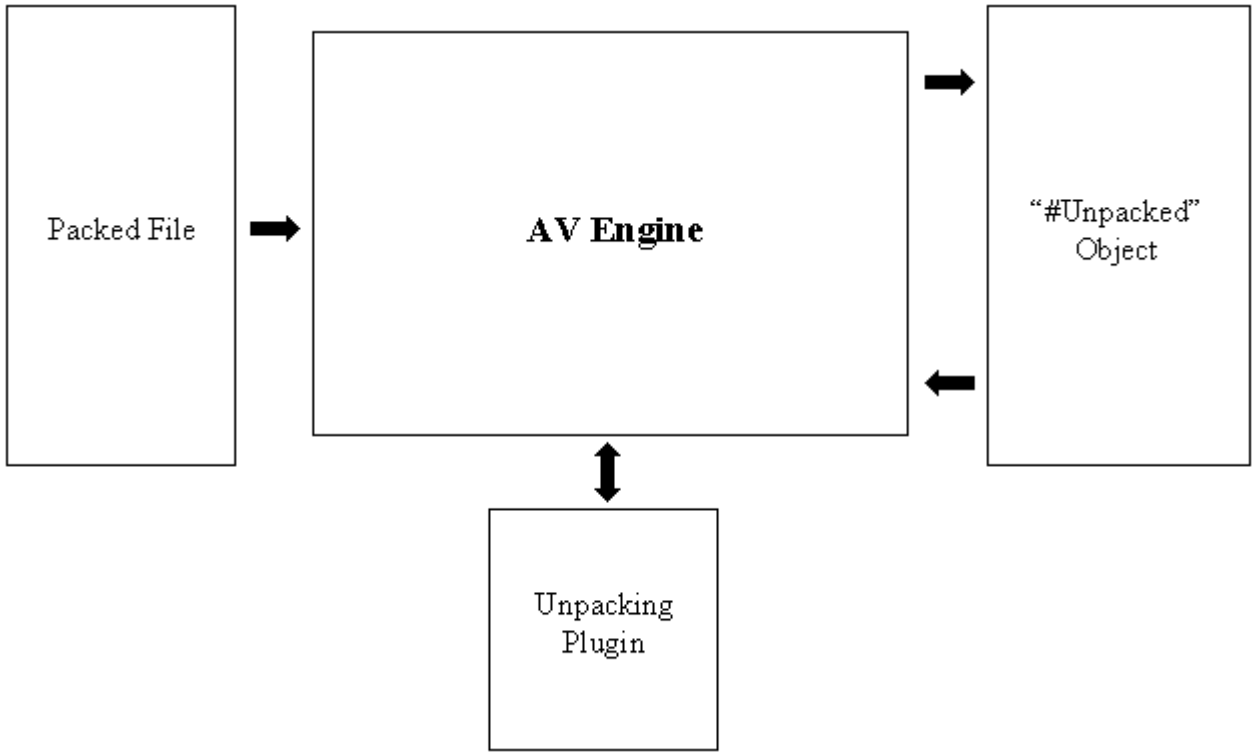


Figure 1: A Schematic of the Unpacking Procedure

In many cases, a packer's functional blocks are not unique. For example, packers tend to share standard, well-tested compression and encryption algorithms. It is only the order of execution of the functional blocks which are unique to an individual packer.

The packer's functional blocks can be executed within the framework of some dedicated pseudo-unpacking code, which communicates with the AV engine as a packer-specific plugin. These plugins may then be released using data updates, to unpack new packers as soon as possible. The release cycle of a data update is much shorter than the full engine update cycle.

In figure 1 the packed file is processed by an unpacking plugin to create an unpacked file (an "#Unpacked" object), which resembles the original unprocessed host in binary content and, perhaps, PE entrypoint. This "#Unpacked" object can be sent back to the AV engine to be processed anew.

Requirements for Hybrid Unpacking

The following stages are required for a file to be unpacked using the plugin framework:

- The engine detects, using generic packer heuristics, that the file may be packed
- The engine decides to present the file to the unpacking plugins
- A plugin decides that it is interested in the file by recognising certain characteristics specific to the packer
- The plugin instructs the engine on how to parse the file
- The plugin extracts the parameters required to decompress or decrypt
- The plugin calls the framework to decompress the file and creates an "#Unpacked" object
- The "#Unpacked" object is fed back to the engine for further processing/scanning

The following prerequisites are necessary to create the unpacking plugin:

- The ability to reverse-engineer packer code to determine its functional blocks
- The AV engine must be able to execute several known, fixed algorithms for decompression, decryption, checksum, etc
- The AV engine must be able to execute simple, variable loops, eg via emulation
- There must be an interface between the AV engine and an external unpacking plugin, which allows the plugin to call the AV engine's fixed algorithms or loop handling on demand. The plugin will need to use the AV engine to execute a packer's functional blocks
- At a bare minimum, for scanning purposes, the stream created after processing the packed file via the corresponding unpacking plugin must have the same binary code content as the

original “host” file before it was packed. Retrieving the correct PE entrypoint for the unpacked specimen would be an added bonus

The hybrid procedure focussing on the unpacking plugin will be described in more detail using MEW 1.1 as an example.

Functional Blocks for MEW 1.1

MEW 1.1 is a mid-range packer in terms of unpacking complexity. The MEW packing utility lists various options to employ in creating the packed version of the host. However, reverse engineering the packer’s code for a typical sample reveals three main functional blocks:

- A version of the APLib decompression algorithm, characterised by the following assembly code snippet with magic numbers:

```
3D 00 7D 00 00          cmp eax, 7D00h

73 0A                   jnb Address1

80 FC 05                cmp ah, 5

73 06                   jnb Address2

83 F8 7F                cmp eax, 7Fh

77 02                   ja Address3
```

  APLib is one of the most common compression algorithms used in packers, eg the packer Packman also uses the same version of APLib as MEW 1.1.

- The LZMA decompression algorithm, characterised by the following assembly code snippet (creating a buffer containing several consecutive occurrences of 04 00 04 00):

```
B9 9B 1B 00 00          mov ecx, 1B9Bh

B8 00 04 00 04          mov eax, 04000400h

F3 AB                   rep stosd
```

  LZMA is used in a handful of packers, most notably Upack and NSPack.

- The LZMAFilter algorithm to retrieve relative calls and jmps (re-patched after decompression) , characterised by the following assembly code snippet:

```
AC                      lodsb

2C E8                   sub al, E8h

74 06                   jz FoundCallOrJmp

FE C8                   dec al

74 02                   jz FoundCallOrJmp
```

```
EB 0B                   jmp GetNextByte
```

Hence, to unpack MEW 1.1 using the hybrid approach, the AV engine must be able to decompress a standard version of APLib (let us call it "APLibVer0") and LZMA. The AV engine should also be able to handle the LZMAFilter algorithm.

MEW 1.1 has several internal table structures, which store necessary information such as the offsets to the compressed data. These offsets will need to be passed to the functions APLibVer0 and LZMA to effect decompression.

MEW 1.1 Plugin

In the plugin model, the bulk of the processing is done in the AV engine. The plugin simply contains instructions for the order of execution of functional blocks in the AV engine.

The MEW 1.1 plugin is written in a language which can interact with the AV engine. In this section a dedicated pseudo unpacking code is used, which is interpreted by the AV engine. Note, the contrived keywords of the unpacking code (called "Dedicated Unpacker Language" or "DUL" hereafter) are meant to be representative only, and do not reflect the language used by our implementation.

```
%PluginName: MEW 1.1%

{

/* Make sure it is MEW 1.1 */

if (!CheckPacker("MEW11")) exit


/* Find pointer to compressed data */

/*

Search for immediate value (I) moved into register esi

Return "I" as a Pointer (P$I) into packed file

*/

TableOffset = FindInstruction("mov esi, P$I")


/* APLibVer0 compressed data is at TableOffset+0Ch */

StartOffset = TableOffset+0Ch


/* Functional Block 1: Decompress using APLVer0 algorithm */

/* There can be several blocks of APLibVer0 compressed data */
```

```
APLLoop:

(DecompressedSize, CompressedSize) = APLVer0(StartOffset)

StartOffset = StartOffset + CompressedSize + 4

DestinationAddress = GetDword(StartOffset - 4)

if (DestinationAddress != 0) goto APLLoop


/* Functional Block 2: Now decompress using LZMA algorithm */

/* After the APLVer0 is complete StartOffset now points to LZMA compressed data
*/

(DecompressedSize, CompressedSize) = LZMA(StartOffset)


/*

By now we should have an object containing most of the host’s binary. All we
need to do is the LZAMFilter to get something that can be scanned

*/


/* Functional Block 3: Now filter fully decompressed data */

StartOffset = ObjectStart(“#UNPACKED”)

NumberOfBytes = DecompressedSize

LZMAFilter(StartOffset, NumberOfBytes)

/*

The host entrypoint virtual address is stored in a table. We can retrieve it,
translate it to a meaningful address in our “#UNPACKED” object, and set the
“ObjectPEEntryStart” to this address to be sent back to the AV engine for
scanning

*/

}
```

DUL Source Code for MEW 1.1 Plugin

The DUL code listing has been simplified to describe the main points in the unpacking routine. Certain details, such as determining when the compression type switches from APLibVer0 to LZMA, have been omitted. Nevertheless, the code accurately describes the method to call decompression routines.

Decompression algorithms usually simply require the starting offset for the corresponding compressed data as a parameter. Unlike decryption loops, decompression algorithms do not require a size (number of bytes) to process. This is because the compressed data usually has signature bytes, which inform the decompression algorithm that it has finished. However, the format of compression is quite rigid, and the starting offset has to be exact for the decompression to flow.

In the DUL source code, the MEW 1.1 assembly needs to be parsed to find the table which determines the starting offset of the first block of APLibVer0 compressed data. The start of subsequent blocks of compressed data, both APLibVer0 and LZMA, has to be calculated by incrementing the original start offset by the number of bytes consumed during each corresponding decompression (ie the compressed size of each block).

It is assumed that the AV engine's decompression algorithms create an "#UNPACKED" object, in memory or in a temporary file, containing the decompressed data. The LZMAFilter has to run over the decompressed bytes, since it is the decompressed bytes which are patched in the first place. Hence the call to LZMAFilter requires a pointer to the "#UNPACKED" stream as a parameter.

Finally, the MEW 1.1 code would need to be parsed again to determine the host entrypoint, ie the PE entry to the underlying original file, if this is indeed required. This entrypoint then needs to be translated to an offset in the "#UNPACKED" stream before sending the object back to the AV engine for processing/scanning.

The Pros and Cons of the Hybrid Approach

Once the infrastructure for the use of DUL code is in place, the unpacking language can be used to unpack different packers.

As mentioned before, many packers share standard algorithms, especially those related to compression. Of course, this means that the AV engine must always carry around certain standard algorithms at all times, but each algorithm typically serves several DUL plugins. Unfortunately, the source code for certain algorithms is not freely available in the public domain, or needs to be translated into a language compatible with the AV engine.

The major advantage in using fixed engine-based algorithms over emulation is the performance enhancement in terms of the speed of unpacking. Decompression algorithms, in particular, consume a relatively large number of instructions and are, thus, very slow to emulate.

Packer	DUL plugin (ms)	Emulation (ms)
LameCrypt (simple decryption)	0.84	20.18
FSG (ApLib decompression)	3.14	283.24
MEW 1.1 (ApLib and LZMA decompression)	17.17	6446.78

Table 1: Performance comparison between Emulation and DUL plugins

Table 1 shows the large differences in unpacking times between using a DUL plugin and emulation (note, the unpacking times shown for emulation are lower estimates) for samples of comparable sizes packed with three different packers. The statistics show clearly that a DUL plugin is much faster than emulation. Furthermore, as the complexity of unpacking increases, the divergence between the relative unpacking times increases markedly.

In addition, emulation is unreliable on packers with relatively complex functionality. Using DUL, any extra functionality in the packer, like anti-debugging tricks, can be ignored while focussing on the core functional blocks of the packer. Emulation, on the other hand, would need to be able to handle the functionality of the packer, which is deliberately designed to make emulation difficult, if not impossible.

However, emulation can have an advantage over the static hybrid method when faced with new or modified packers. The DUL code can only be written after careful analysis of the packer's code, which can be time consuming. This means that a DUL plugin is unlikely to be effective on packers which have not been seen before.

In addition, DUL code may not be possible for packers that use complex algorithms which are not already available in the AV engine.

Some malware, like members of the W32/Bagle and W32/Sober families, have been known to use packers unique to themselves which vary significantly from one sample to the next. Given the effort required, DUL plugins are not appropriate for some of these unique packers, which may be sample-specific, never to be used again. Once again, emulation would be advantageous in this scenario.

Notwithstanding, for most new packers, DUL code can be written reasonably quickly and delivered with data updates without any major changes to the AV engine. Once written, the DUL plugins tend to be fast and are extremely effective on packers which are common across several families of malware. The plugins are not only useful for scanning purposes. They can also be used to create files for static analysis, although in this case extra coding may be required to retrieve the host's import table.

Conclusion

The use of packers to obfuscate malicious code is a real problem. A lot of malware in the wild would already be detected but for the layer of packing. Therefore, it is essential to counter this problem by unpacking the sample to recreate a minimal portion of the original host file.

There are several different approaches to an unpacking solution, ranging from highly inflexible, but quick, static algorithms in an AV engine to highly flexible, albeit slow, emulation. The hybrid solution attempts to achieve the centre ground by encompassing the best of both worlds.

However, there is no perfect solution on the horizon. All methods of unpacking have their advantages and disadvantages, and they are all susceptible, to a degree, to problems caused by new, more complex packers. The hybrid method based on DUL plugins has the potential to be improved by extending the language and implementing more standard decompression/decryption algorithms into the AV engine. DUL plugins can be deployed at short notice, providing the advantage of a

quick reaction to packer variations without major changes to the AV engine or large increases in the memory footprint.

It is clear that flexibility and performance (speed) are mutually exclusive in the unpacking domain. We believe that the hybrid approach to unpacking strikes the right balance between speed and flexibility.

References

Shipp, A. (2004).Unpacking strategies, Virus Bulletin Conference, Chicago, USA.

Stepan, A. E. (2005). Defeating Polymorphism: Beyond Emulation, Virus Bulletin Conference, Dublin, Ireland.

Graf, T. (2005). Generic unpacking – How to handle modified or unknown PE compression engines, Virus Bulletin Conference, Dublin, Ireland.

Zlib web page – http://www.zlib.net

Aplib web page – http://www.ibsensoftware.com/products_aPLib.html

LZMA SDK web page - http://www.7-zip.org/sdk.html

Szor, P. (2005). The Art of Computer Virus Research and Defense, Chapter 6 and Chapter 7: Symantec Press, ISBN 0-321-30454-3.

Marinescu, A. (1999). ACG in the Hole (p 8-9), Virus Bulletin (July issue).

Ferrie, P. & Szor, P. (2001). Hunting for Metamorphics, Virus Bulletin Conference, Prague, Czech Republic.

Natvig, K. (2001). Sandbox technology inside AV scanners, Virus Bulletin Conference, Prague, Czech Republic.

Natvig, K. (2002). Sandbox II: Internet, Virus Bulletin Conference, New Orleans, USA.

Vulnerabilities of the usage of digital signature

Ferenc Leitold
University of Veszprem - Veszprog Ltd., Hungary

About Author(s)

Ferenc Leitold graduated from Technical University of Budapest in 1991. He received his Ph.D. at Technical University of Budapest too, in 1997 in the theme of computer viruses. Currently he teaches in the Department of Information Systems at University of Veszprém. He teaches computer programming, computer security, and computer networks. His research interest is based on computer viruses: mathematical model of computer viruses, automatic methods for analysing computer viruses. According to the CheckVir project (www.checkvir.com) of Veszprog Ltd. he is dealing with testing and certification of anti-virus software products.

Vulnerabilities of the usage of digital signature

Abstract

The efforts made concerning the use of the digital signature represent a significant step in the information technology. These efforts do not or just to a small extent concern the issues of security. Unfortunately, there are many devices whose use for digital signatures raises serious problems concerning security.

When we sign a document on paper we rely on our eyes and it depends on our mental ability whether we can make sense of what we can see. Our eyes will give evidence to the fact that the signature is put only onto the document that we intend to sign.

When we prepare an electronic signature we must believe our eyes and we must make sense of it. Also, we must believe that the information displayed on the screen (or on any other output device) corresponds to a raw of bits stored in the memory or the mass storage, which has exactly the same sense as we interpret it. We must believe that the unit constructing the signature (e.g. an outer card reader connected on a serial port) provides only that raw of bits with the electronic signature whose correspondence is displayed on the screen.

From what I have said above we can claim that when we use a multipurpose computer to prepare electronic signatures then we must completely trust its hardware and software installation and the proper operation of the software. Of course, we cannot check this in any visible objective way. Thus, for a potential attacker there are two opportunities:

- *affect the presentation,*
- *manipulate the signing procedure.*

This paper highlights that there are a lot of security problems of the digital signature usage. It will be demonstrated using some samples.

Introduction

The tram drivers in Budapest often warn the passengers: “Attention! There are pickpockets in the tram, take care of your properties!” Who would question the rightfulness of the driver’s warning? Every public message warning of attacks has two effects: on the one hand the people hold their bags tighter and put their papers in their inner pockets—it is just natural for law-abiding people. On the other hand, the message heels the attention of the potential pickpockets for the good opportunity that should be exploited now or other times. This article attempts to present the possible points of attack relating to the use of the digital signature. Of course, this warning may also have two effects. Nevertheless, it is in the interest of everyone using or accepting digital signatures to be aware of the dangers when they are using the system and they should do their best to limit the number of these gaps. If somebody gets on the tram, they too would like to be aware of the dangers.

The aim of this paper is to examine the security of the computers relating to the digital signature and to explore the potential points of attack. These issues are closely related to the general security problems of the computers. The first part of the article covers the theoretical background of the operation of the electronic signature. The following chapters deal with the security problems of using a personal computer for providing and checking digital signatures and the analysis of the security issues of forwarding digital signatures.

Theoretical bases

To understand the security problems it is essential to overview the mathematical theoretical bases, which are to provide the security of the digital signatures.

Public Key algorithms

The basic idea of the public key methods *(Ralph Merkle, 1974)* is that the keys used for encoding (enciphering) and decoding (deciphering) need not necessarily be the same.

The public key algorithms are similar to a padlock that has two keys—one for opening it and another for closing it. The two keys are closely linked together and they are called a *'keypair'*. The key that closes the padlock (encodes the message) is called *'public key'*, and the one that opens the padlock (deciphers the message) is called *'cipher key'*. If the public key is made known then anybody can send us a ciphered message. Of course, to prevent the unauthorised parties from having access to the ciphered message the cipher key must be kept in secret. Formally, the encoding and the decoding can be expressed as

$$M=C_e(m) \text{ and}$$
$$m=D_d(M)$$

where m is the original message, M is the encoded message, C_e is the encoding algorithm using the public key e, and D_d is the decoding algorithm using the cipher key d.

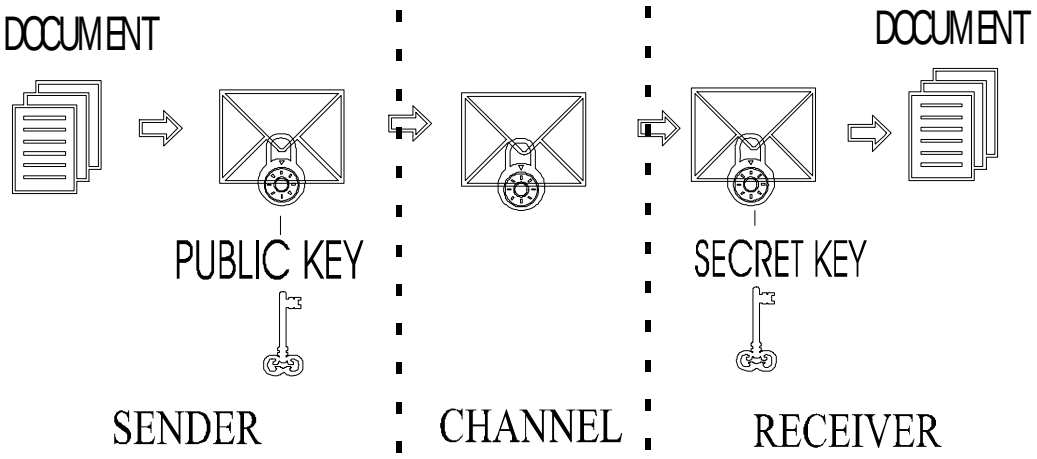


Figure 1 Encoding and decoding with public key algorithm

If the roles of the keys can be swapped over, that is

$$M=C_d(m) \text{ and}$$
$$m=D_e(M),$$

then message *M* can be decoded with public key *e*, therefore message *M* carries the information the cipher key of which public key was used for ciphering i.e. validating it.

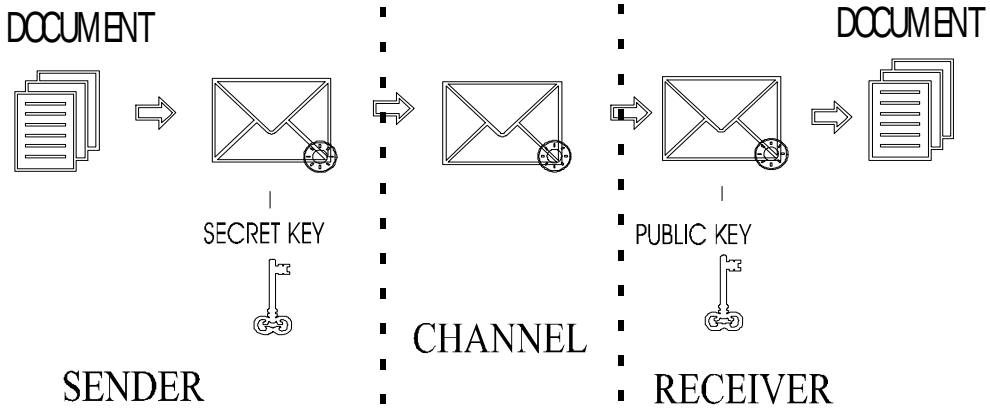


Figure 2 Validation with public key algorithm

The encoding procedure can be applied two times: for validating with our own cipher key and for ciphering with the public key of the receiver. Formally:

$$M=C_{e2}(C_{d1}(m)) \text{ and}$$
$$m=D_{e1}(D_{d2}(M)),$$

where *d1* and *d2* are the cipher keys of the two parties and *e1* and *e2* are the public keys of the two parties. Thus the public key algorithms provide solution to the problems of both validation and ciphering.

Digital signatures

The algorithm of the digital signature uses the algorithm of the public key encoding described above. This method is described in Figure 3. The first step is that the document to be signed appears on the computer of the sender; from the binary code series of the document the fingerprint peculiar to the document is prepared. This can be carried out with the help of the Hash algorithms. This fingerprint is then ciphered with the cipher part of the keys of the public key algorithm. The code series prepared this way is the **digital signature** rendered to the document. Afterwards the sender forwards the document and the digital signature rendered to it. The receiver receives the document and the digital signature and does the following: with the help of the same Hash algorithm he prepares the fingerprint rendered to the document. Also, from the digital signature, he prepares the fingerprint rendered to the digital signature by using the sender's public key. If the two fingerprints are identical, he can make sure that the digital signature was made with the cipher pair of the public key used for the supervision. The mathematical theory of the method does NOT ensure that the digital signature has been rendered to the person signing the document or that the digital signature has been made with the knowledge of the owner of the cipher key.

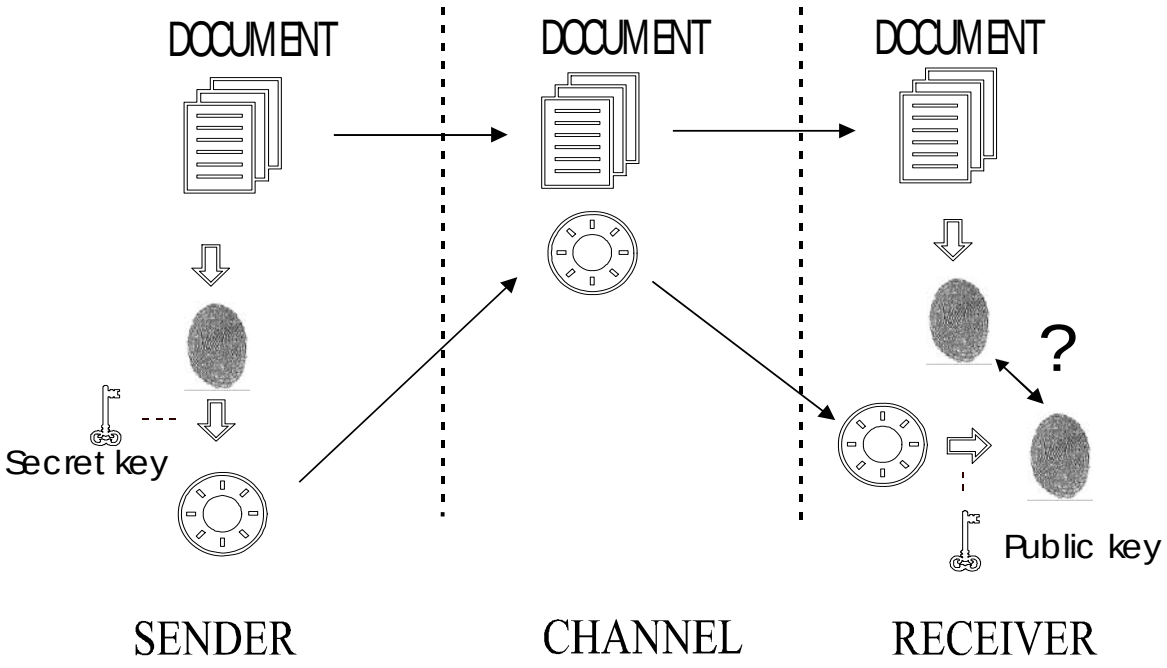


Figure 3 The operation of the digital signature

Traditional signature – electronic signature

When we sign a document on paper we rely on our eyes and it depends on our mental ability whether we can make sense of what we can see. Our eyes will give evidence to the fact that the signature is put only onto the document that we intend to sign (Figure 4.).

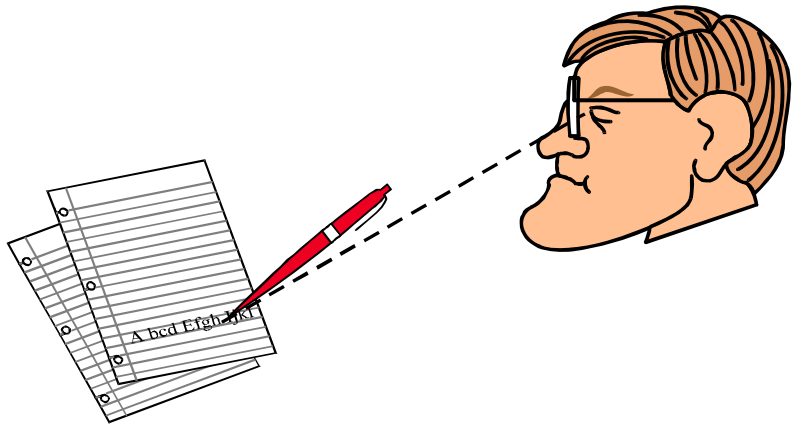


Figure 4 Traditional signature

When we prepare an electronic signature we must believe our eyes and we must make sense of it. Also, we must believe that the information displayed on the screen corresponds to a bit series stored in the memory or the mass storage, which has exactly the same sense as we interpret it. We must believe that the unit constructing the signature (e.g. an outer card reader connected on a serial line) provides only that bit series with the electronic signature whose correspondence is displayed on the screen (Figure 5.).

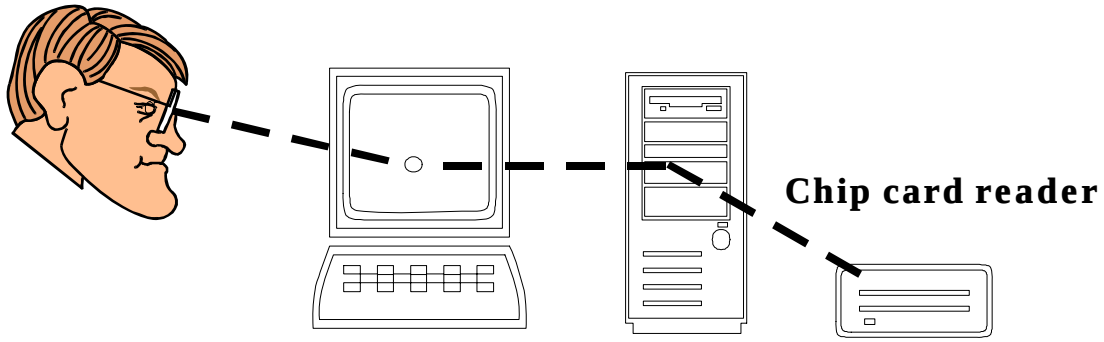


Figure 5 Electronic signature

From what I have said above we can claim that when we use a multipurpose computer to prepare electronic signatures then we must completely trust its hardware and software installation and the proper operation of the software. Of course, we cannot check this in any visible objective way. Thus, for a potential attacker there are two opportunities:

- to affect the presentation,

- to manipulate the signing procedure.

Affecting the presentation

We ought to expect the document to be signed to contain all the information to interpret and present it. If, for the interpretation, information from another source is required then the presented image of the document can be affected. Such typical information is the image of the characters. Word and certain PDF documents do not contain the fonts required to present the image of a document. Thus, by changing the fonts the image of the document will be different in a new environment. Unfortunately, the ASCII text files are not different either. Despite the fact that there are no fonts here we must know the image of the characters for the presentation. And this is information outside the document (the bit series of the text), which is fixed by the ASCII standard, but the presentations are made by the hardware and software of the computers. In the case of a VGA card the image of the characters can be overwritten! The problem is NOT relating to the operation system. Under the terminals of Linux and UNIX systems exists the notion of fonts too and a StarOffice document does not contain the images of the letters either. To ensure the correspondence between the document and its presented image it is indispensable that the document **in itself** contains the binary images of the characters.

The question is what possibilities an attacker has to exploit the gap in the security system:

- If the attacker can have access to another computer he can change the image of the letters in any of the fonts. These applications are freely accessible on the Internet.
- He can change the letters with a small program of his own too.
- He can send this program even in e-mail. For this there are tremendous amounts of viruses sent by e-mail today.

We can claim now that a malicious attacker can easily—especially if his partner does not know much about security in information technology—enter a program into his lay partner's computer, which then ensures that the signer will see something different from what he intends to sign. He can also eradicate himself entirely from the layman's computer following the signing, e.g. at a definite time. After that the lay user would try to prove his truthfulness in vain; the electronic signature is approved evidence in the courts of many countries…

Manipulating the signing procedure

If we are fully aware of what we intend to sign — or at least we believe so — we can provide the document with our electronic signature. In order to do this we need a *signature-making device*. This device contains software as well as hardware elements. When we make up our minds to sign a document this device ensures that every condition is fulfilled to carry out the signing procedure without any further interaction. If we use some kind of a chip-card, after inserting the card every condition is given for the signature. We cannot check manually whether we render our signature to that particular bit series and we cannot make sure that there is no signature rendered to other bit series.

A typical possibility for attack could be the following method: a small program, which is entered into the computer this way or the other watches an interactive activity that the user must carry out after satisfying every condition for the signature—e.g. he has entered the chip-card into the reader. The

program senses what information the user program sends to sign for the card reader. The program sends this to the reader and waits for the response but it does not forward it to the user program but sends another bit series for the reader to sign. When this has happened, the program then sends back the signed answer to the user program. All this happens so quickly that the user does not notice anything. This 'small program'—like the majority of e-mail viruses—can even send the signed bit series back to the attacker with its own SMTP routine.

Using documents with digital signature

The advantage of the digital signature is that the two signers of a common declaration (e.g. a contract) need not meet. It is enough to exchange the electronically signed declarations (electronic documents) through messages. Nevertheless, everybody prefers to handle their contracts discreetly and would not like any unauthorised party to have access to them. PKI offers an excellent opportunity to avoid this by ciphering the messages.

Forwarding on the Internet

The electronically signed document must be sent to the receiving party. We can do this through data media or by mail. In any of the cases the solution is not less comfortable than in the case of the traditionally signed paper document. The only significant difference is that forwarding through data media we can send or carry the information ciphered. With paper documents this cannot be done.

A natural way of forwarding a document is sending it through the Internet. Sending a message on the Internet is just as safe as sending information on a postcard, therefore it is essential to cipher the document.

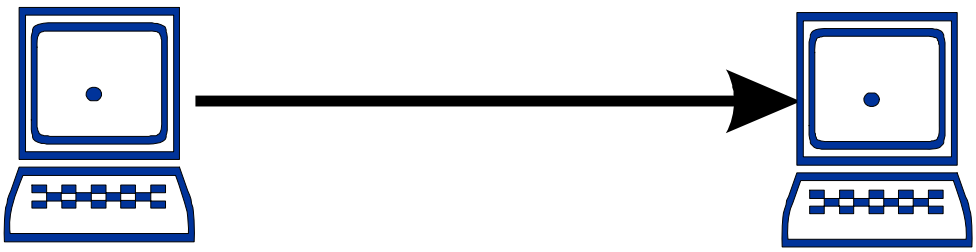


Figure 6 Forwarding on the Internet

Forwarding through firewall

Today every business or institute has internal information infrastructure or inner network. It is also inevitable to build a firewall at the meeting point of the internal network and the Internet. The firewall must watch the traffic between the inner network and the Internet and by influencing it tries to protect the internal network from the dangers coming from the Internet. A well-configured firewall system must have packet filtering devices and content filtering possibilities (e.g. virus protection) too.

Let us assume that the two managers intend to exchange their electronically written documents on the Internet. The most natural way to do it is to send the signed message through e-mail. It is essential for both of them to keep the contents of the document in secret even in the internal network. Therefore they cipher their messages, which can be easily made with the PKI technology (Figure 7.).

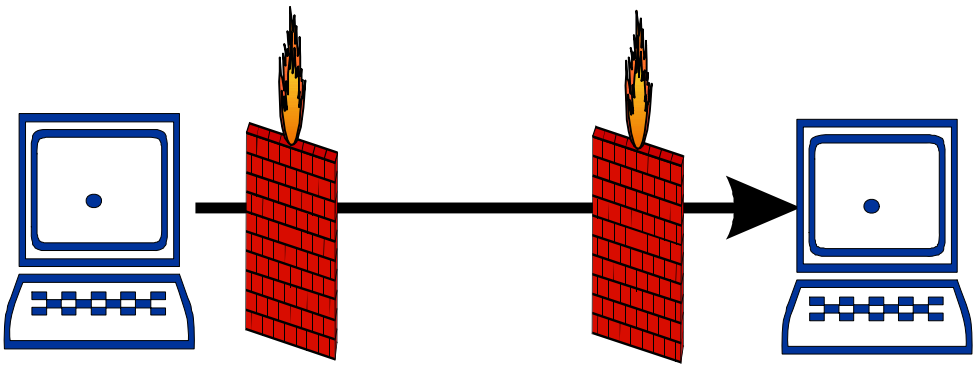


Figure 7 Forwarding through firewall

But the real security gap occurs at the firewall. The system managers maintaining the firewalls have two options:

- They configure the firewall so that it does not allow the documents through the contents of which they cannot check. But in this way the ciphered and signed documents will never get through to the other party.
- The firewall is set to let through the ciphered messages without supervision. Then the two managers can exchange the signed and ciphered messages. But this 'security relief' is just enough for an attacker to enter through the firewall an attacking program, which is ciphered with the manager's public key (Figure 8.).

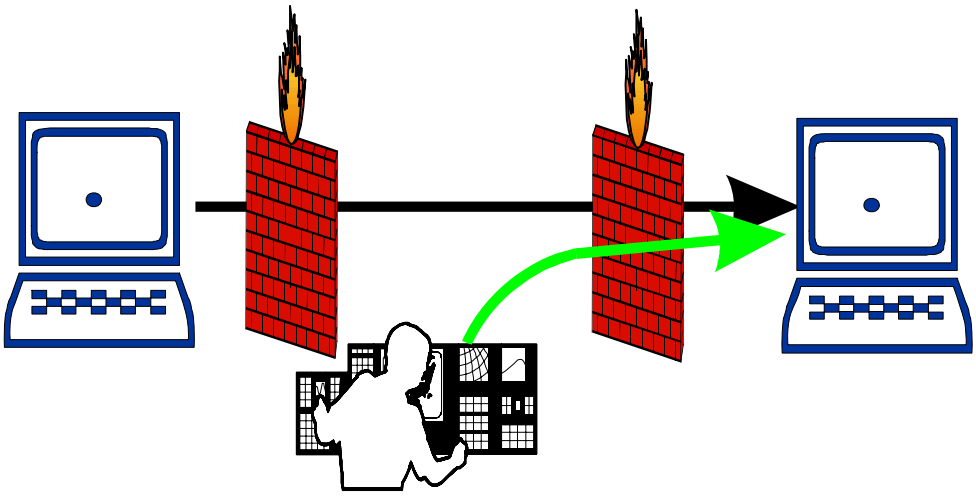


Figure 8 Opportunity to attack through the firewall

Suggestions

Knowing the above described security problems we can claim that the use of the digital signature is not perfectly safe. If a single or limited purpose target machine—a bank ATM or mobile phone—is at our disposal then we can use the achievement of the digital era entirely safely. If we assume that the signature is made on a computer that is used for other purposes as well, we must face serious security problems. It does make a difference what we sign, or rather, what we sign can be interpreted only in the way we mean it. It does make a difference what system and what device we are using for the signature. And, finally, it does make a difference for what purpose we intend to use the signed document and how we intend to forward it.

One of the basic shortages of the legal regulations in many countries is that they do not formulate unambiguously the circle of data that can be signed electronically. The regulation ought to define 'text' and 'letter'. Which are the electronic forms that can be regarded as 'text communicated with letters'? Is a scanned A/4 page saved in a binary picture form acceptable if it contains letters only? It is also a basic expectation that the regulated forms and standards should be made public and accessible for all, otherwise how could we supervise a document based on a ciphered form? The supervision could be assisted with supervising software with an open source code.

A further demand is that regulations should be disposed about the forwarding of the documents signed digitally. At the moment a few of the regulations about the digital signature excludes the use of the keys for other—i.e. ciphering—purposes. The ciphering keys could be classified similarly to the keys used for the signatures. With a common regulation the providers of the authentication could a lot more easily carry out both authentications than separately.

If possible, I find it necessary to inform the users about the potential sources of danger. This could be one by the providers of authentication because they know the devices used for digital signatures thus they ought to provide the guidelines about the secure use. Also, the users ought to be trained about the security precautions and the protection.

The users—private individuals, business enterprises or public institutions—are interested in the secure operation of their systems. The security of the machines used as computers as well as signature-making devices is closely related to the overall security of the computer. Making the computer more secure—with a firewall or virus protection or security regulations—the making of the digital signature becomes more secure too. There is no solution to the security of the digital signature that could be detached from the overall security of the computer.

Summary

The fact that four times as much money is being spent on the security of the computers as three years ago proves that we are facing more and more sources of danger, which ought to be handled by the computer users with greater care.

The electronic signature—rendering it to the document—raises a number of security problems when we use a computer for making signatures. The reason is that there is no operation system,—probably

there cannot be any—which could provide sufficient security for making digital signatures at the moment. The users must obtain a **security culture**, which can help to prevent the potential problems and, if there has been trouble, to reconstruct the system.

References

[1] DIRECTIVE 1999/93/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 13 December 1999 on a Community framework for electronic signatures Official Journal of the European Communities, 19.1.2000 *http://www.ict.etsi.org/*

[2] Community framework for electronic signatures *http://europa.eu.int/scadplus/leg/en/lvb/l24118.htm*

[3] Internet X.509 Public Key Infrastructure Certificate and CRL Profile RFC, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 1999 *http://www.ietf.org/html.charters/pkix-charter.html*

[4] Internet X.509 Public Key Infrastructure Certificate Management Protocols RFC, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 1999 *http://www.ietf.org/html.charters/pkix-charter.html*

[5] Internet X.509 Public Key Infrastructure Representation of Key Exchange Algorithm (KEA) Keys in Internet X.509 Public Key Infrastructure Certificates RFC, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 1999 *http://www.ietf.org/html.charters/pkix-charter.html*

[6] Internet X.509 Public Key Infrastructure Time Stamp Protocol (TSP) Internet Draft, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 2000 *http://www.ietf.org/html.charters/pkix-charter.html*

[7] Simple Certificate Validation Protocol (SCVP) Internet Draft, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 2000 *http://www.ietf.org/html.charters/pkix-charter.html*

[8] Alternative Certificate Formats for PKIX-CMP Internet Draft, Internet Engineering Task Force (IETF) Public-Key Infrastructure (X.509) (pkix) Working Group, 2000 *http://www.ietf.org/html.charters/pkix-charter.html*

[9] Public-Key Cryptography Standards RSA Laboratories *http://www.rsasecurity.com/*

[10] Recommendation X.509 (03/00) - Information technology - Open Systems Interconnection The directory: Public-key and attribute certificate frameworks ITU Telecommunication Standardization Sectuion (ITU-T) *http://www.itu.int/itudoc/itu-t/rec/x/x500up/x509.html*

[11] Information and Communications Technologies Standards Board (ICTSB) *http://www.ict.etsi.org/*

[12] European Electronic Signature Standardization Initiative (EESSI) *http://www.ict.etsi.org/eessi/EESSI-homepage.htm*