Malwares as Interactive Machines: A New Framework for Behavior Modelling -Extended Abstract-

Grégoire Jacob[1/2], Eric Filiol[1], Hervé Debar[2]

[1] French Army Signal Academy,
Virology and Cryptology Lab., Rennes, France
eric.filiol@esat.terre.defense.gouv.fr

[2] France Télécom R&D, Caen, France
{gregoire.jacob|herve.debar}@orange-ftgroup.com

March 9, 2007

1 Introduction

By making a survey on the different techniques of behavioral detection, we have quickly noticed that a multitude of systems exist, each one redefining its own behavior model. An underlying idea is then to provide a reference model for expressing these malicious behaviors, detached from the nature of the detection systems for interoperability sake. In a first place, we thought that Turing Machine equivalent languages were a good starting point, since most of abstract virology models and semantic detectors likewise, rely on Turing Machine equivalent formalisms. But along our work, we have gradually become aware that some dynamic notions such as interactions and concurrency were fundamentally missing in order to apprehend certain new malicious trends. By essence, malwares, as adaptive and resilient agents, are likely to use these mechanisms intensively. In this paper, we extend the malware models to the Interaction Machine formalism. This theoretical model is particularly adapted to apprehend these missing notions.

In a first part, we are thus going to describe the known lacks of Turing Machines and equivalent models. In the following one, we will introduce the extended model we have chosen as solution: Interactions Machines. According to this model, we will provide new definitions for interactive and distributed malwares. We will also describe more precisely different classes of interactions and study their impact on the detection complexity. The second part of the paper is less theoretical and aim to provide a model framework based on interactive languages, specifically designed for describing malicious behaviors. To complete our study and assess the relevance of this framework, we provide applied descriptions for several behaviors usually used by current malware strains.

2 Shortcomings of the Turing Machine models

Since the first formal works by Cohen in the eighties [1], very few formal works on malware models have been published. Its original approach based on Turing Machines has been superseded in more recent publications by more expressive representations able to take into account additional concepts such as mutations and stealth. L. Adleman was the first to introduce a new model based on the theory of recursive functions [2] which has been extended afterwards by Z. Zuo et M. Zhou [3]. More recently G. Bonfante, M. Kaczmarek and J.-Y. Marion have put forward a significant formalism based on the foundation of computability which matches up with the previous models [4]. Unfortunately, as stated by the Church-Turing thesis, all these models rely on formalisms eventually equivalent to the Turing Machines.

On the one hand, these models have proved effective in capturing important concepts like replication, mutation or stealth. Most important of all, they have provided fundamental results on the detection complexity. On the other hand, P. Wegner rightly underlines the fact that Turing Machines remain undoubtedly insufficient to model open systems such as modern computers [5]. Extending the formalism of replicating virus to more complex malwares will eventually fail because of important missing dynamic concepts:

- **Interactions:** Open systems frequently interact with the external world. These interactions can be seen as ways to import and export data between external adversaries and the machine. These remote adversaries remain out of control of the local machine and thus can not be captured by the Turing formalism. Typical examples would be a network connection with a remote machine or simply user intervention.

- **Parallelism and concurrency:** Simple parallelism between processes can be addressed by Turing Machines. On the contrary, concurrency or distributivity, traducing strong dynamic interactions between the processes, can not. This result have already been established in previous publications [6][7]. Distributivity to evade detection is a real threat and is already deployed in certain malwares as mentioned in E. Filiol's recent paper on k-ary malwares [8][9].

Only one related work has already tried to extend the viral models to take interactions into accounts. To achieve this, F. Leister has introduced a new mathematical formalism based on Random Access Stored Program Machines with Attached Background Storages [10]. These storage facilities are in fact additional bands with concurrent access in reading and writing modes shared by all the processes. This paper was a first step towards considering interactions, but dynamic concepts still misse like unconstrained interactions and non-determinism. We aim to introduce a richer model based on Interaction Machines addressing these limitations. In particular, our contribution provides:

- Interaction classes according to the nature of the adversary and channel,
- A formal definition for interactive and distributed malicious strains,
- New detection complexity results,
- An interactive language to operationally model behaviors,

- Criteria to assess the model soundness and completeness,
- Description of several behaviors to give hints of the language possibilities.

3 Interaction Machine based models

Interactions Machines are one of the several alternative extensions of Turing Machines put forward in order to address their limitations. According to the definition advanced by P. Wegner [11], an interaction machine can be described as a Turing Machine with dynamic input and output facilities. Basically, an Interaction Machine has the same expressive power than a Turing Machine with oracles and infinite input [12]. The main interest is that an oracle can hypothetically solve any problem even undecidable and manipulate data of infinite size. With regards to Interaction Machines, the oracle can model the behavior of any adversaries of the machine taking the time and interaction history into account. The input of the oracle model the data sent during interactions and the output, the data received. In case of unilateral interactions, either the input or the output can be null.

Based on this theory and the formalism introduced by G. Bonfante et al. [4], we will provide definitions for two new classes of viruses:

Definition 1. Our definition of an interactive virus is based on the definition of an implicit virus. The performed actions depend on interactions with adversaries whose results can be taken as inputs. Let $C_1, ..., C_k$ be k semi-computable disjoint subsets of a computation domain D, $V_{1,1}, ..., V_{i,j}$ be $i \times j$ semi-computable functions and $\Phi^1, ..., \Phi^n$ be the n oracles associated to n interactive adversaries. An interactive virus v exists such that, for all p and x, the equation is satisfied:

$$\varphi_v(p,x) = \begin{cases} V_{1,1}(v,p,x,\Phi^1(v)) & \text{if} \quad (p,x,\Phi^1(v)) \in C_1 \\ ... & \\ V_{n,k}(v,p,x,\Phi^n(v)) & \text{if} \quad (p,x,\Phi^n(v)) \in C_k. \end{cases}$$

Definition 2. A distributed malware is made up of two or more programs executing and interacting to achieve malicious behaviors. Distributed malwares according to our definition are in fact a subset of the k-ary malwares defined by E. Filiol [8][9]. Let Φ be an oracle reflecting the interactions of two programs. The programs v and w are components of a distributed virus if there is a semi computable function f satisfying the equation:

$$\varphi_{v,w}(p,x,y) = f(\varphi_v(p,x,\Phi(v,w)), \varphi_w(p,y,\Phi(v,w))).$$

The definition has then been extended to distributed viruses over n components using a graph representation and a more complex system of equations.

Interactions may be different according to the entities put into relation. By considering the different classes of interactions, we will be able to associate an equivalent complexity to the oracles modelling them. We have therefore defined three main classes:

- (Class I_1) Interactions with inert objects (files, registry keys,...) whose complexity is linear,

- (Class I_2) Interactions with active objects through defined interfaces (network sockets, mutex,...) whose complexity is NP-complete,
- (Class I_3) Free interactions with active objects (concurrent processes, ...) whose complexity is undecidable.

The oracle complexity is then multiplied by a combining factor depending on the network of mutual interactions between objects. Finally, we have reached the following result with regards to the detection complexity:

Proposition 1. The set of interactive (resp. distributed) malwares for a given propagation function is respectively $\Pi_2^{\Sigma_0}$, $\Pi_2^{\Sigma_1}$ and *Undecidable* according to the class or interaction considered.

4 A formal semantic based on interactive machines for malware behaviors

This theoretical background justifies the importance of interactions and their impact on the detection. Based on Interaction Machine formalism, we have established a semantic of malware behaviors adopting an object-oriented approach. The formal grammars have the advantage of providing a better understanding of the malware effects with great manipulation facilities while remaining enough formal for a high level representation. The malware is basically seen as an object with internal attributes and mechanisms. Additional interfaces are then provided for interaction with external objects. These objects have been classified according to their particular purpose from the malware perspective. The whole grammar is too voluminous to be written down in this abstract but we will give hints of it through examples in the next part. Nevertheless we can briefly give a few examples of the object classes we have defined:

- Permanent objects able to survive a complete system reboot (files, registry keys, etc),
- Temporary objects existing for a determined time over a system session (events, mutex, etc),
- Boot objects providing the malware facilities to execute automatically its code (run registry key, service descriptor table, etc),
- Communicating objects providing the malware facilities to propagate to remote systems (network connections, P2P shared folders, drivers, etc),
- The autoreference to the malware itself.

With regards to internal mechanisms, the language put forward is sound and complete since it is Turing Complete. Regarding communications, the interaction machines extend the Chomsky hierarchy of languages to the domain of non-computable functions. As stated by P. Wegner, interaction languages require additional dynamic listening and transmitting operators as well as operators for non-deterministic choices [11]. Consequently, these three types of operations have been integrated. Relating to interactions, the grammar is obviously sound since the object-oriented approach is inspired from reality. On the

other hand, completeness can not be proved formally. We can simply mention the fact that the grammar is able to describe the three kinds of interactions we have previously defined, both synchronously and asynchronously.

5 Behavior modelling through interactions

In order to assess our model and measure further its completeness, we have chosen to confront it to existing malwares. To do so, we have proceeded to a behavior survey for several representative malicious strains thanks to information gathered from malware observatories or analysis published by experts. Doing so, we have identified different techniques used to achieve several classes of typical malicious behaviors. We have then described these behaviors as subgrammars of the generative one. Only the replications mechanisms will be introduced here. Even if they are not shown in this short abstract, we have also established descriptions for the following behaviors: polymorphism, metamorphism as rewriting rules, overinfection and activity tests, residency, stealth, emulation detection and proactive defence.

Example 1. The first replication method is simple duplication where no target is required to host the code. The code is first stored in a local buffer traduced by the generic variable V_{code}. It is then stored in a newly created permanent object O_{clone}. During the duplication, mutations can occur even if they are not described within this abstract.

$V_{code} \in var$
$O_{clone} \in obj\_perm$

$$
\begin{array}{lll}
(i) & < Duplication > ::= & < Creation >< Reading > \\
 & & \quad < Mutation >< Writing > \\
 & & | < Reading >< Creation > \\
 & & \quad < Mutation >< Writing > \\
(ii) & < Creation > ::= & create \quad O_{clone}; \\
(iii) & < Reading > ::= & receive \quad V_{code} \leftarrow this; \\
(iv) & < Writing > ::= & send \quad V_{code} \rightarrow O_{clone};
\end{array}
$$

Example 2. Contrary to duplication, infection requires an existing entity to host its code. As a consequence, the first phase of the replication always consists in crawling in the system to look for a potential target. In order to describe a valid target, conditions modelled by C_{valid} are defined on the nature of the target, one of them being to be not previously infected. This description takes into account append and prepend modes of infections, whether destructive or relocating the original code buffered in the variable V_{save}.

$V_{target}, V_{code}, V_{save}, V_{comparison} \in var$
$C_{valid} \in const$
$O_{target} \in obj\_perm$

$$
\begin{array}{lll}
(i) & < Infection > ::= & < Searching >< Opening >< Relocating > \\
 & & \quad < Reading >< Mutation >< Writing > \\
 & & | < Searching >< Opening >< Reading > \\
 & & \quad < Relocating >< Mutation >< Writing >
\end{array}
$$

$$
\begin{array}{lll}
(ii) & < Searching > & ::= while(V_{comparison} := (\neg(= (V_{target}, C_{valid})))\{ \\
& & \quad open \quad O_{target}; \\
& & \quad receive \quad V_{target} \leftarrow O_{target}; \\
& & \} \\
(iii) & < Opening > & ::= open \quad O_{target}; \\
(iv) & < Relocating > & ::= receive \quad V_{save} \leftarrow O_{target}; \\
& & \quad send \quad V_{save} \rightarrow O_{target}; \\
& & | \ \epsilon \\
(v) & < Reading > & ::= receive \quad V_{code} \leftarrow this; \\
(vi) & < Writing > & ::= send \quad V_{code} \rightarrow O_{target};
\end{array}
$$

Example 3. Propagation is a third way of replicating more specific to worm. Contrary to the two previous cases of local replication, propagation is the capacity to replicate over remote systems. The code is no longer copied in a permanent object but rather sent to a communicating object. According to the nature of the channel used, a formatting phase may be required. For example, mail propagation requires the construction of a mail structure with valid headers and the code of the malware attached encoded in a base 64 format. Notice that encoding the malware code may take several passes.

$V_{code}, V_{formatted}, V_{parameter}, V_{position} \in var$
$C_{header}, C_{hsize} \in const$
$O_{channel} \in obj\_com$

$$
\begin{array}{lll}
(i) & < Propagation > & ::= < Opening >< Reading > \\
& & \quad < Mutation >< Transmitting > \\
& & | < Reading >< Opening > \\
& & \quad < Mutation >< Transmitting > \\
(ii) & < Opening > & ::= open \quad O_{channel}; \\
(iii) & < Reading > & ::= receive \quad V_{code} \leftarrow this; \\
(iv) & < Transmitting > & ::= send \quad V_{code} \rightarrow O_{channel}; \\
& & | < Formatting > \\
& & \quad send \quad V_{formatted} \rightarrow O_{channel}; \\
(v) & < Formating > & ::= V_{position} := (\&(V_{formatted})); \\
& & \quad [V_{position}] := (C_{header}); \\
& & \quad V_{position} := (+(V_{position}, C_{hsize})) \\
& & \quad < Encoding > \\
& & \quad [V_{position}] := (V_{code}); \\
(vi) & < Encoding > & ::= V_{code} := (< Op2 > (V_{code}, V_{parameter})); \\
& & \quad < Encoding > \\
& & | \ \epsilon
\end{array}
$$

6 Conclusion and perspectives

Along this paper we introduce a reference semantic based on interactions in order to describe malicious behaviors. The theoretical approach adopted during the first parts justifies their importance and measures their heavy impact on the detection complexity. The operational approach of the second parts provides a relevant framework with respect to our behavior survey. In order to achieve a greater completeness, the scope of the survey should be increased to a wider

range of malwares. Anyhow, the generative grammar proves to be sufficiently generic to integrate additional behaviors and the object classification can be easily refined without deep modifications.

Working at a high level of representation, as it is the case with formal languages, has several advantages. It proves really useful in expressing the final aim of behaviors rather than the techniques used to achieve it. Moreover this semantic brings into light functional similarities more evolved than simple instruction equivalence which is the major drawback of most current behavior-based detection systems. Eventually, it could be worth considering integrating our framework to existing semantic analysis systems for malware detection as in [13][14] and [15].

References

[1] F. Cohen, *Computer Viruses*. PhD thesis, University of South California, 1986.

[2] L. M. Adleman, "An abstract theory of computer viruses," in *CRYPTO '88: Proceedings on Advances in cryptology*, pp. 354–374, 1990.

[3] Z. Zhihong and M. Zhou, "Some further theoretical results about computer viruses," *The Computer Journal*, vol. 47, no. 6, pp. 627–633, 2004.

[4] G. Bonfante, M. Kaczmarek, and J.-Y. Marion, "On abstract computer virology from a recursion theoretic perspective," *Journal in Computer Virology*, vol. 1, no. 3-4, pp. 45–54, 2006.

[5] P. Wegner, "Why interaction is more powerful than algorithms," *Communications of the ACM*, vol. 40, no. 5, pp. 80–91, 1997.

[6] R. Milner, "Elements of interaction: Turing award lecture," *Communications of the ACM*, vol. 36, no. 1, pp. 78–89, 1993.

[7] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag New York, Inc., 1992, ISBN:0-387-97664-7.

[8] E. Filiol, *Techniques Virales avancées*. Springer, IRIS Collection, 2007, ISBN:2-287-33887-8.

[9] E. Filiol, "Formalisation and implementation aspects of k-ary (malicious) codes," *Journal in Computer Virology*, vol. 3, no. 3, EICAR 2007 Special Issue, V. Broncek Ed., 2007.

[10] F. Leitold, "Mathematical model of computer viruses," in *Best Paper Proceedings of EICAR*, pp. 194–217, 2000.

[11] P. Wegner, "Interactive foundations of computing," *Theoretical Computer Science*, vol. 192, no. 2, pp. 315–351, 1998.

[12] P. Wegner, "Interaction as a basis for empirical computer science," *ACM Computing Surveys*, vol. 27, no. 1, pp. 45–48, 1995.

[13] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, "Semantic-aware malware detection," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 32–46, 2005.

[14] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, "Detecting malicious code by model checking," *Lecture Notes in Computer Science*, vol. 3548, pp. 74–187, 2005.

[15] J. Shin and D. Spears, "The basic building blocks of malware," tech. rep., University of Wyoming, 2006.