Formal Model Proposal for (Malware) Program Stealth

Éric Filiol[*]

Army Signals Academy
Virology and Cryptology Laboratory,
Rennes (France)
`eric.filiol@esat.terre.defense.gouv.fr`

Abstract. Recent advances in stealth techniques have dramatically increased the malware hazard. More recently, virtualisation-based rootkits like SubVirt or BluePill have strongly challenged the classical capabilities of malware detection. In this paper, we formalize stealth and rootkits technologies in a far different way that those typically considered, that is to say as a more or less complex set of hooking and kernel subversions. By comparing stealth or rootkits to steganographic techniques, we propose a new information theoretic-based formalisation that enables to define the problem of stealth detection in a more powerful and high level way than the existing ones. Consequently, it yields new perspectives of what detection of stealth really is and how to address the relevant problem on a practical basis. In particular, this modelling gives clues and potential practical approaches to detect the most recent rootkit techniques like SubVirt or BluePill.

Keywords: Stealth, Virtualisation, Steganography, Information theory, Malware, BluePill, SubVirt, Rootkits, Detection.

1 Introduction

As soon as malware writer has begun to produce malware, they tried to make them more or less invisible for the user and later, with the evergrowing antiviral detection capabilities, with respect to the antivirus. Hiding in fake corrupted cluster, hooking interruption or APIs... have been very soon considered for the general purpose of *stealth*. Present malware still use these classical techniques and will do. And contrary to recent claims [19, 20], the game never really changed and stealth approaches fundamentally remain the same. Even the term of *rootkit* is nothing but an extension of stealth and could be itself defined as a more or less complex set of stealth techniques.

However, for most IT security professionals, recent advances in stealth – and particularly virtualization-based stealth [19] – seem to represent an important

[*] Also Associate Senior Professor at ESIEA - Laval *filiol@esiea.fr*

and significant step and consequently strongly challenge the word of classical security. What about the computer security gloomy future forecast by some recent claims? Fortunately not even if – as it has been previously the case, for example with respect to polymorphism – existing protection and detection techniques are bound to evolve.

Another key misperception comes from the fact that the concept of stealth is frequently misunderstoood with that of malware. Stealth must should be seen only as a set of techniques whose aim is to protect a file or a process against detection in the broadest sense. The file or process to be hidden are not necessarily malicious ones. The best example is probably the Sony Rootkit, even if we can consider this approach as a rather trivial and unefficient one. But other cases are bound to emerge.

All this misperception comes from the fact that there does not exist any universal definition – independant of the operating system, the software or the hardware – of what stealth really is. The only existing formalisation attempt [22] considers a subclass of stealth only. As far as stealth classification is concerned, the Rutkowska's recent attempt [20] has completely mistaken the different concepts behind stealth while totally neglecting the computability and complexity issues.

In this paper we propose such a universal model for stealth based on information theory and steganography. Let us precisely define the latter concept

Definition 1 *(Steganography and steganalysis) The* steganography *is the set of techniques which not only enable the security of the information – COMSEC (COMmunication SECurity) aspect – but also and above all the security of the (information) tranmission channel – TRANSEC (TRANSmission SECurity) aspect. The* steganalysis *is the set of detection techniques whose purposes is to detect the use of steganography and to access the hidden information.*

This definition gives an obvious parallel between stealth and steganography: the COMSEC aspect is related to the malware itself (its code and its actions) while the TRANSEC aspect relates to the malware execution and interactions with the target system. It becomes thus interesting to develop this parallel in the context of information theory and statistics in order to completely model stealth. This is the central topic of this paper. This approach moreover enables to show that if malware detection is a general undecidable problem [3, 4], there is no such things as total malware weapons or total invisibility for programs.

The paper is organized as follows. Section 2 presents the our formal model for stealth while Section 3 deals with the formalisation of stealth detection. In order to illustrate and give clues on how detection should evolve to manage the most sophisticated techniques, Section 4 presents some practical aspects of detection before concluding in Section 5.

We will not recall the different existing stealth techniques whether classical or very recent like virtualisation-based ones. The reader will refer to [4, Chap. 7] or [8, 14, 19, 20].

2 Formal Model of Stealth

2.1 Previous Works

There is only one known attempt of formalisation with respect to program stealth. It has been introduced by Zuo and Zhou (2004) by means of recursive functions.

Let Ω be an enumerable set whose elements describes either a program or a data. Let be $\sigma : \Omega \rightarrow \mathbb{N} \cup \perp$ a total recursive function which describes a formal abstraction of a computer system resources (executable programs, operating system, system calls, clock, memory, disks...). If we have $\sigma(x) = \perp$ then the function σ is undefined with respect to x (hence the function is indeed total). Let us denote a function whose value at n is y and which is the identity function for the other values, by the symbol $\sigma[y/n]$. For conciseness purposes, we will use the expression $\sigma[v(n^{\sigma})]$ instead of expressions like $\sigma[v(\sigma(n))/n]$, which means the value in σ corresponding to name n has been performed an operation v in it.

Definition 2 *(Stealth virus) (Zuo & Zhou, 2004) Let v be a total recursive function and $\mathfrak{S}$ a recursive function denoting a system call. Then the pair $(v, \mathfrak{S})$ is a* stealth virus *with respect to $\mathfrak{S}$ if there exists a recursive function h such that, for all i,*

$$\phi_{v(i)}(\sigma) = \begin{cases} D(\sigma), & \textit{if } T(\sigma) \\ \phi_i(\sigma[v(S(\sigma)^{\sigma}), h(\mathfrak{S}^{\sigma})]), & \textit{if } I(\sigma) \\ \phi_i(\sigma), & \textit{otherwise} \end{cases}$$

and

$$\phi_{h(\mathfrak{S})}(x) = \begin{cases} \phi_{\mathfrak{S}}(y), \textit{ if } x = v(y) \\ \phi_{\mathfrak{S}}(x), \textit{ otherwise} \end{cases}$$

where the recursive predicates $T(\sigma)$, $I(\sigma)$ represent the payload trigger condition and the infection condition, respectively. The recursive function $S(\sigma)$ is called the selection function.

Whenever $T(\sigma)$ is true, then the payload $(D(\sigma))$ is executed. If predicate $I(\sigma)$ is true, then the virus selects a target program to infect by means of the selection function $S(\sigma)$, infects it and then execute the host program. But a stealth virus will additionally modify or use some system calls or resources in such a way that whenever the system or the user are checking whether an infection occured or not (whatever may be the detection technique used), no infection is detected at all while indeed there is one (even if the antivirus in place used to detect it if no stealth were used).

2.2 Modelling Stealth: Steganography and Information Theory

As presented within the introduction, steganography ains at hiding the existence of a transmission channel. Now, the concept of stealth as usually perceived in computer virology, essentially consists in hiding into a computer system – we will compare it to the transmission channel – both data (program code as an example) and actions directly or undirectly related to the data (during the program execution). According to our parallel, those data and actions will be compared to the message to be hidden.

That first parallel immediately enables to define the *camouflage* techniques as the subclass of trivial stealth techniques which can be assimilated as the steganography applied to "non active" data – a program code in a non execution context – while the true stealth will be defined as the steganography applied "active data": the code/data must be hidden during either their own execution (process hiding) or the system operation to detect them (e.g. file hiding).

There exist several attempt at formalizing steganography. While Hopper, Langford and von Ahn [9] have based theirs on complexity theory, Cachin [1] has considered information theory as a formalisation tool. We will use this second approach to develop our parallel between steganography and program stealth.

Let us first recall the main concepts underlying steganography and let us draw our parallel further.

- a *covertext* C is an innocent looking data in which a *secret* message M may be hidden. A covertext with an embedded hidden (secret) information M is called a *stegotext* denoted S. In our context, C corresponds to the computer system files, data structures and processes that may be used by a (malware or not) program to hide its own code, data and actions;
- the hiding process is performed by means of a key-dependant or not hiding algorithm. Existing stealth techniques generally do not consider any key. However, for generalisation purposes, a key K can be considered as part of the program data;
- the adversary of the communication tries first to guess whether there is some stegotext S within a given population of covertexts – this is equivalent to the classical antiviral detection problem – and secondly to access the message M – this can be compared to the antiviral eradication problem. The general approach is to consider statistical models $\mathcal{P}_{\mathcal{C}}$ and $\mathcal{P}_{\mathcal{S}}$ which describe the populations $\mathcal{C}$ and $\mathcal{S}$ of the covertexts and the stegotexts respectively.

The last item clearly shows that stealth detection may be modelled as a statistical testing problem as presented in [5]. We develop this in Section 3.

From a theoretical point of view, a steganography system (or stegosystem for short) can be redefined as follows (from [1]) by considering the theory developped in [5].

Definition 3 *(Steganography system) Let $\mathcal{P}_{\mathcal{C}}$ and $\mathcal{P}_{\mathcal{S}}$ be distributions on a set $\mathcal{C}$ of covertexts C and on a set $\mathcal{S}$ of stegotexts S respectively. A* stegosystem *is a triplet of probabilistic polynomial-time algorithms (SK, SD, SE) with the following properties.*

- *The* key generation algorithm SK *takes as input the security parameter n and outputs a bit string K (the* stegokey*).*

– *The* steganographic embedding algorithm SE *takes as input the security parameter* n*, a covertext* C*, the key* K *and a message* $M \in \{0,1\}^l$ *to be hidden. It outputs an element* S *of* $\mathcal{S}$*. The algorithm may access both distributions* $\mathcal{P}_{\mathcal{C}}$ *and* $\mathcal{P}_{\mathcal{S}}$*.*
– *The* steganographic extraction algorithm SD *takes as input the security parameter* n*, the key* K *and an element* U *from* $\mathcal{C} \cup \mathcal{S}$*. It will output either a message* $M \in \{0,1\}^l$ *or the special symbol* ? *when either an extraction error occurs or when no message is embedded in* U*.*

For all key K *output by* SK *and for all* $M \in \{0,1\}^l$*, the probability*

$$P[SE(n, K, SD(n, K, M) \neq M] \text{ (stegosystem reliability)}$$

must be negligible in n

From that definition, the parallel between steganography and stealth becomes obvious. Of course this parallel is a broad generalisation of what are or may be all stealth techniques: most of the existing (detectable) stealth mechanism do not use any key K yet (and thus no algorithm SK). They do not use probabilistic algorithm yet as well. The main interest of our model is to clearly show that existing stealth techniques just still represent a trivial subset of all possible stealth techniques. As a consequence, we can forecast and identify more sophisticated techniques which will appear in the future and thus pro-actively consider the defense. On the extreme, the combination of purely steganographic techniques with classical stealth techniques will be inevitably considered by malware writers.

The other interest of our model lies in the fact that detecting the existence of a secret communication channel at least (the TRANSEC aspect) enables the defender to react and protect even if it remains impossible to access the embedded message (the COMSEC aspect). While in a communication context, we can at least jam or cut off the channel, in an antiviral protection context we may at least quarantine the system.

From our model, let us now address the problem of stealth security, that is to say the ability to more or less resist detection.

2.3 Security of Stealth

This of course the most important property to be realized[1]. By considering the taxonomy presented in [1] and the detection formal model in [5], we can formalize the concept of security as far as stealth techniques in the broadest sense, are concerned.

[1] Two other desirable properties could be also considered: robustness and capacity. Robustness could be defined as the ability to resist to system modification. A coding theory approach enables to formalize this property Capacity describes the amount of information that can be embedded into a given system. Information theory can be then again used. Both properties have been addressed in [6].

Let us denote *DSys* the distribution of all possible files, structures and processes of a system that can be used as covertext and *DStealth* the distribution of files, structures and processes that have been effectively used with respect to a given stealth technique. By denoting $\mathcal{P}_Q(x)$ as the probability of x with respect to the distribution Q, we have:

Definition 4 *A stealth sustem is said to to ϵ-secure against a passive attack if and only if*[2]

$$D(P_{\text{DSys}}||P_{\text{DStealth}}) = \sum_{x \in Q} P_{\text{DSys}}(x) \log \left(\frac{P_{\text{DSys}}(x)}{P_{\text{DStealth}}(x)} \right) \leq \epsilon.$$

where Q denotes the space of possible measurements. If $\epsilon = 0$ then the stealth system is said to be perfectly secure.

In other words, the security of a stealth system can be defined as the relative entropy $D(P_{DSys}||P_{DStealth})$ between the two distributions *DSys* and *DStealth*. We have $\epsilon = 0$ whenever *DSys* and *DStealth* are identical.

Let us precise that our woorking context is that of passive attacks, that is to say when the detection analysis does not modify the system being analysed. That corresponds to the traditional case of antiviral detection. But if we consider "active" analysis – the detection modifies the distribution *DSys* and eventually the distribution *DStealth* as well –, we have to consider such a system modification Y. The previous definition forces to no longer take the probabilities P_Q but the conditional probabilities $P_{Q|Y}$. But the model essentially remains the same.

Definition 4 just proposes a theoretical model for stealth in its broadest sense. But it does not tell how to practically measure or even characterize a signficant distribution between the two distributions. It is worth mentioning that the distribution *DStealth* will be a priori unknown while defining the distribution *DSys* may be very complex (see Section 3). As far as practical detection is concerned, the essential interest of Definition 4 lies in the capability to directly apply theoretical and practical results/techniques presented in [4, 5].

When again considering the taxonomy introduced by C. Cachin [2] and when transposing it to stealth techniques, the latter can be organised according the three following classes, which are defined with respect to their capability to remain more or less undetected.

- *Unconditionnally secure stealth.* Then we have $\epsilon = 0$. These techniques may be compared to Shannon's perfect secrecy [4, Chap. 7]. This implies that detection is not possible even with unbounded time and computing resources. It seems very unlikely that classical stealth techniques would ever belong to this class without using some kind of cryptography and/or classical steganography. This remains an open problem. However by comparing unconditional security with undecidable computability, the concept of testing simulability as defined in [4, 5] gives clues of how stealth techniques belonging to this class, could be designed;

[2] The notation log() in this paper refers to the base 2 logarithms.

– *Statistically secure stealth.* The adversary is an arbitrary unbounded algorithm (time and computing) and has a negligible advantage[3] $\epsilon = \mathcal{O}(\frac{1}{n})$.
– *Computationnally secure stealth.* The adversary is an arbitrarily probabilistic, polynomial-time algorithm and has a negligible advantage $\epsilon = \mathcal{O}(\frac{1}{n})$.
– *Unsecure stealth.* The adversary is a deterministic polynomial time algorithm. Moreover ϵ does no longer depend on n. It could be somehow considered as a trivial subset of the previous class.

Existing stealth techniques – including rootkits and virtualisation-based techniques like *SubVirt* [14] or *BluePill* [19] – belong to the last class (Unsecure stealth). The lack of security for all these techniques comme from the fact that system modification for stealth purposes are such that only the knowledge of the distribution *DSys* is required for the detection. In other words, as soon as there exist an estimator for which the system distribution significantly differs from *DSys*, we can infer that there is something wrong and that some stealth is likely to be used.

As far as *BluePill*-like techniques are concerned, they are likely to belong to the class of unsecure stealth, contrary to the claim of their author [19]. It is very likely that some estimator exists in order to detect any such techniques (e.g. when modifying *PageFile* files). Moreover behaviour based detection could be trigger when the program precisely switches into a stealth state. Some detection aspect of *BluePill* techniques will be addressed in Section 4.

Lastly, the absence of cryptographic techniques directly determines the unsecurity of stealth according to the Kerckhoffs cryptographic law that states that no system security can rely on the system secrecy alone [11]. That is why system modifications (during the boot sequence as an example) could be hidden in the future by using secret keys and security parameters in order to consider stealth techniques in "less insecure" classes.

3 Formalisation of Stealth Detection Techniques

3.1 Previous Works

As far as stealth detection is concerned, there is only one known (theoretical) results (Zuo & Zhou, 2004). If we denote D_s the set of stealth viruses and D_s^{fixed} the set of stealth viruses sharing the same kernel[4].

Theorem 1 *The set D_s^{fixed} is Π_2-complete while the set D_s is Σ_3-complete.*

This results refers to the class of computationnally secure stealth only. It clearly shows that detecting such stealth programs has a huge complexity that is far beyond the detection complexity of most other malware. The use of stealth for

[3] That is precisely the utility of the security parameter n to make in such a way that $\lim_{n\to\infty} \epsilon = 0$.
[4] The kernel of a virus is the set of the functions/predicates $T(\sigma)$, $I(\sigma)$, $D(\sigma)$ and $S(\sigma)$.

anti-antiviral purposes is finally an efficient strategy for a malware writer, if and only if stealth belongs to suitable classes and is, of course, efficiently implemented. On the defense side, on the contrary, being aware of those principles is an unquestionable requirement when having system protection in charge. In both side, it is important to stress on the fact that if total detection is impossible (Cohen, 1986; Filiol & Josse, 2007), total stealth is also an impossibility. So there is a least one good news.

3.2 Antiviral Detection and Statistical Testing

As shown in [5], all existing antiviral detection techniques can be modelled as a statistical testing. Any detection process consists in deciding whether an infection has occured (hypothesis $\mathcal{H}_1$ or *alternative hypothesis*) or not (hypothesis $\mathcal{H}_0$ or *null hypothesis*). Both hypotheses are described by their respective distribution (in the context of stealth, *DStealth* and *DSys* respectively). i

The main basic tool to build a test is the *estimator*, denoted E. Its observed value computed on a given sample is denoted e. According to the hypotheses of the testing, E has a different probabilistic law. So by comparing the value $E = e$ with that of a *decision threshold* T, we keep or reject $\mathcal{H}_0$. This threshold in fact partitions the set of possible values for E in two disjoint sets of $\mathbb{R}$, denoted A (*acceptance region*) and $\overline{A} = \mathbb{R}\backslash A$ (*rejection region*).

It is worth mentioning that the choice of estimators is generally *a priori* dictated by *DSys* (detection of unknown viral or stealth techniques) unless some *a posteriori* knowledge on the techniques can be used (detection of already known techniques).

But any decision is marred with two kind of errors summarized in Table 1.

Decision	$\mathcal{H}_0$ true	$\mathcal{H}_1$ true
Keep $\mathcal{H}_0$	$1-\alpha$	β
Reject $\mathcal{H}_0$	α	$1-\beta$

Table 1. Probabilities of Error Attached to Statistical Testings

In other words, we have:

– the type I error α given by:

$$\alpha = P[E \in \overline{A}|\mathcal{H}_0 \text{ true}] = P[e > S|\mathcal{H}_0 \text{ true}],$$

and in an antiviral detection context, if the null hypothesis describes the fact that no infection occurred, the type I error represents the false positive probability.

– the type II error β given by:

$$\beta = P[E \in A|\mathcal{H}_1 \text{ true}] = P[e < S|\mathcal{H}_1 \text{ true}],$$

In an antiviral context, it corresponds to the non-detection probability (if $\mathcal{H}_0$ is the non infection hypothesis).

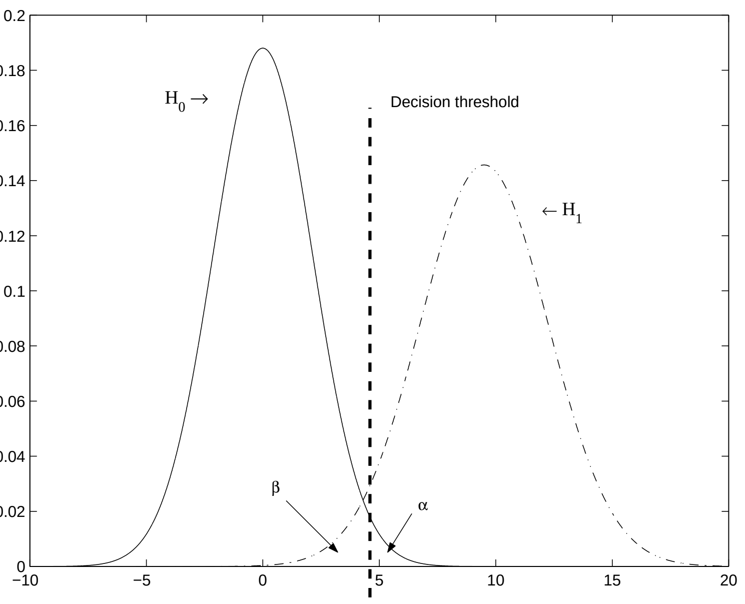

Fig. 1. Statistical Model of Antiviral Detection

These two different errors are described in Figure 1. It is essential to point out the fact that these two error risks are opposite to one from the other [5]. Moreover, in most practical testings, the type I error is privileged. Thus we have to know the law of the null hypothesis. On the contrary, the law of the alternative hypothesis is generally unknown (we cannot forecast which viral techniques has been used). That implies that determining the exact value of β is most of the time impossible.

To end with statistical testing, it is essential to stress on the fact that error risks α and β are, by definition never equal to zero, except if the $\mathcal{H}_0$ and $\mathcal{H}_1$ probabilistic law domains (density functions) are disjoint. But in the latter case, it becomes obvious that deciding is easy and that no testing is required at all. The different cases of detection are discussed in [5].

3.3 Formalisation of Stealth Detection

From our model given in Section 2 and from Section 3.2, we can now formally address the problem the stealth detection. For that purpose, we will consider the concept of binary relative entropy between two distributions (namely *DSys* and *DStealth*). It is may be defined by the following equality

$$\varDelta(\alpha, \beta) = \alpha \log\left(\frac{\alpha}{1+\beta}\right) + (1-\alpha)\log\left(\frac{1-\alpha}{\beta}\right).$$

This equality is established by using the Neyman-Pearson lemma [15, page 65]. A equivalent approach can be found in [4, 5]. The concept of binary relative entropy enables then to use results presented in [1] and to transpose it to stealth detection.

Theorem 2 *In stealth system that is ϵ-secure against passive detection, the non detection probability β and the false positive probability α satisfy*

$$\Delta(\alpha, \beta) \leq \epsilon.$$

If the distributions *DSys* and *DStealth* are equal then $\Delta(\alpha, \beta) = 0$ and the stealth system belongs to the class of unconditionnally secure stealth. The reader will not that our present formalisation directly refers to the class of statistically secure stealth. All the problem in detecting stealth lies in finding at least a suitable estimator for which distributions *DSys* and *DStealth* differs up to the error risks. We will consider this aspect in Section 4.

3.4 A New Definition of Stealth

The previous formalisation enables now to give a more suitable, general definition of stealth by considering the powerful concept of testing simulability. Beyond the intrinsic risk of wrong decisions during antiviral detection, there exist a much worse situation: when the attacker (e.g. the malware writer) use the detection techniques against the defender (the antivirus). As soon as the first one knows which tools are used by the second, he is able to perform what we call *statistical testing simulability.*

Let us give a definition for this concept introduced in [4, 5].

Definition 5 *Simulating a statistical testing consists for an adversary, to introduce, in a given population $\mathcal{P}$, a statistical bias that cannot be detected by an analyst by means of this testing.*

There exist two different kinds of simulability:

- the first one does not depend on the testings (and their parameters) the defender usually considers. It is called *strong testing simulability.* In a stealth detection context, strong simulability exist when the malware writer, who has identified any of the techniques used by one or more antivirus, designs new stealth techniques that cannot be detected with respect to the target antivirus;
- on the contrary, the second one does depend on those testings that the attackers aims at simulating. It is called *weak testing simulability.* Bringing weak simulability into play is somehow tricky. It requires to get a deep knowledge of the testings to be simulated. The central approach consists in introducing a new property P' in such a way that the estimator E in use remain in the acceptance region of the testing (generally that of the null hypothesis). Let us recall that during the decision step, the tester checks whether $E < S$

or not. Thus weak simulability consists in changing the value $S - E$ while keeping it positive. For that purpose, we use the intrinsic properties of the relevant sampling distribution.

A number of practical examples of testing simulability have been presented in [4]. Moreover, this concept enabled to give a statistical equivalent to the Cohen's undecidability result of antiviral detection [3].

The interest of testing simulability lies on the fact that stealth can be defined as the capacity of defeating detection and remaining undetected by using testing simulability. In other words, stealth consists in simulating the distribution *DSys*. A strong consequence is that if such non detection successfully occurs (up to the statistical risks) it is only possible with respect to a given testing and a given estimator. But it is intuitively impossible to design stealth techniques in order to simulate all possible testing and taking the infinite set $\mathcal{E}$ of all possible estimators E. Current research aims at proving this claim on a mathematical basis. In particular, if some "secret" detection was used, it would be impossible for the malware writer to simulate it.

As a conjecture, we may suppose that the class of unconditionnally secure stealth techniques can be defined with respect to known detection techniques only. This means that absolute stealth does not exist at all. In other words, it is impossible to introduce stealth into a system without significantly modifying the distribution *DSys* for some estimators. All the antivirus expert's work consists in finding an efficient enough estimator.

4 Practical Aspects of Stealth Detection

These aspects essentially rely on the ability to find some efficient enough estimator in order to discriminate the distribution *DSys* against the distribution *DStealth*. In this last section, we will illustrate this aspect with a few practical example of estimators. As a preliminary, it is essential to keep in mind that the concept of statistical estimator must be taken in the broadest sense: it is the result of any function computed on a sample drawn from a system under analysis. Consequently, it can relates either to sequence-based aspects (from a simple character string to complex, dynamic bytes structures), behaviour-based aspects or system activities.

4.1 Detection from Inside the System

Many differentr techniques have been proposed by antivirus researchers in order to detect stealthy programs from inside a corrupted system. In particular, some of the most recent techniques are dedicated to discriminate instructions executed within a virtual environment against those executed in a physical system. Due to the lack of space, we will not list them but the reader will refer to the study performed in our lab [10] for a detailed survey of these techniques.

In order to illustrate our model on a practical basis, let us detail how the *Patch Finder* rootkit detection [18] operates. This techniques monitors API hooks. In this particular case, the estimator E is the average number of CPU instructions executed during the invocation of several API functions. On a clean system and for a given system call s, let us assume that $E = m_s$ (mean value for the distribution *DSys*) while on a corrupted system, we will have $E = m'_s$ (mean value for the distribution *DStealth*). If the function s has been hooked, it is expected that $m_s < m'_s$.

But recent advances of stealth techniques make use of virtual machine like in *SubVirt* [14] or of thin hypervisor like in *BluePill* [19]. Then the problem of detection becomes far more complex since the detection technique may be subverted and manipulated by the stealth mechanisms which modify the estimator value in such a way that we decide the distribution *DSys* as the valid one. This is the reason why we have proposed in Section 3.4 a new definition for stealth based on testing simulability. This comes from the fact that as soon as an estimator is known to be efficient at detecting a given stealth technique, malware writer adapt by simulating the testing. In this context, the recent evolution of virtalization capabilities make detection far more complex.

Let us now consider the technique developped by Cédric Lauradoux [12, 13]. His seminal work is based on the Popek and Goldberg [16, 7] formalisation work of virtualization and on the improvement given by Robin and Irvine [17]. Two of the most essential properties required for an efficient hypervisor are given hereafter.

Property 1 *A process which is executed under the control of an hypervisor must behave similarly as if this process would be directly executed in a physical environment.*

Property 2 *(Case of hardware-based virtualization [17]) There must a way to automatically signal the VMM when a VM attempts to execute a sensitive instruction. It must also be possible for the VMM to simulate the effect of the instruction. Sensitive instructions include:*

- *instructions that attempt to change or reference the mode of the VM or the state of the machine;*
- *instructions that read or change sensitive registers and/or memory locations such as a clock register and interrupt registers;*
- *instructions that reference the storage protection system, memory system, or address relocation system. This class includes instructions that would allow the VM to accee any location not in its virtual memory;*
- *all I/O instructions*

As far as Property 1 is concerned, there exists an exception: the *execution time.* This comes from the fact that an hypervisor has to perform virtualization-specific operations would not exist in a physical system. Moreover, the hypervisor has to manage the interrupts caused by the execution of sensitive instructions (e.g. instructions that affects the behaviour of the physical machine) within virtual machines [12].

Property 2 may also be violated. Interrupt mechanisms in x86 architecture are not efficient enough to systematically verify this property. As an example, a set of 17 instructions x86 (including SGDT and SIDT instructions) [17] are violating it. This demonstrates that the x86 architecture is not well-adapted to implement virtualization.

From all that, a test based on the violation of Property 1 has been proposed by C. Lauradoux [12] improving a technique a technique initially proposed by Rutkowski [21]. The core principle relies on the fact that the execution time of some particular operations on virtual machines significantly differs from the execution time with respect to physical machines. Let us consider the following program

```
X = (float *) &pageX[offsetX];
Y = (float *) &pageY[offsetY];

time = HardClock();
memcpy(X, Y, 512);
time = HardClock() - time;
```

Figure 2 presents the execution patterns obtained on PowerPC (G3 and G4) processors and with the PearPC emulator (wich emulates the PowerPC instruction set). When measuring the access time to array elements, we can observe periodic anomalies to the different processor cache memories. Our estimator E thus maybe the number of such anomalies. Whenever an emulator is active, E will be closer to zero than when instructions are executed on a physical machine. Thus to detect an active emulator, we can stress a suitable component of the micro-architecture and measure the processor response times.

C. Lauradoux has developped [12, 13] other more sophisticated techniques to detect every kind of virtual machines. The approach essentially remains the same and all the work aims at finding some efficient estimator. In particular, he has extensively analysed the execution times of a loop according to the different possible scheduling policies under Linux (SCHED_OTHER, SCHED_FIFO, SCHED_RR.

4.2 Detection from Outside the System

Any of the detection methods used from inside a system corrupted by stealth program is bound to fail by simply manipulating the estimator and its environment (e.g. the system clock) and by simulating the associate testing.

The only possible detection approach is consequently to detect from aouside the system in order to have a distribution DSys that cannot be simulated by the stealth program since it can simply be not accessed by it. For that purpose, whatever may be the external estimator, the distribution *DSys* will be measured on a clean different system. As for corrupted systems, they will be marred with a distribution DStealth.

Methods under current development in our laboratory [10] consist in considering instructions I whose average execution time in a physical machine (accessing

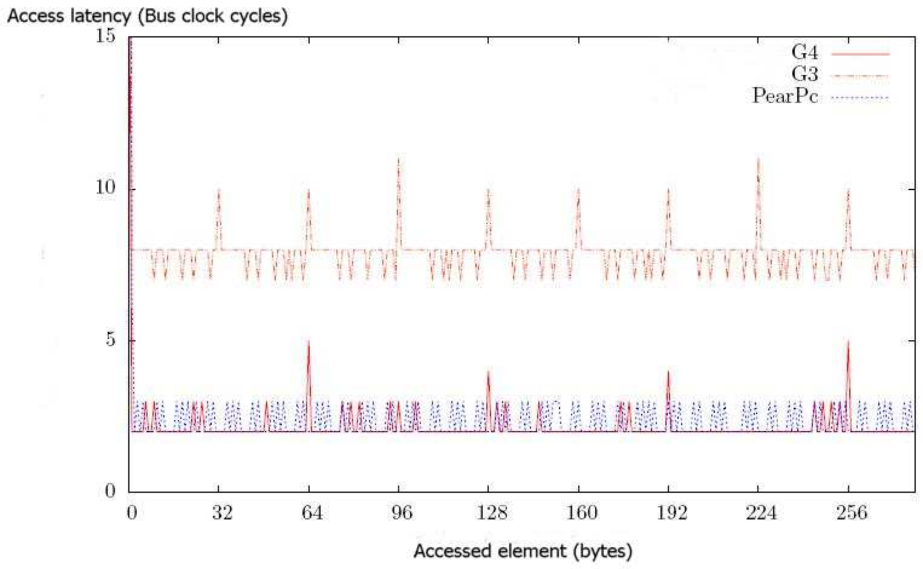

Fig. 2. Comparison of execution anomaly periods between an emulator and a physical system

the hardware) is T. Property 1 being violated in the corrupted system, execution time T' will always be such that $T' = T + \delta > T$. Indeed, any stealth action (manipulation) is bound to introduce a delay. Moreover the time reference being external, the stealth program cannot manipulate it.

Of course, the delay δ is generally small enough to forbid any significant decision. The solution is then to consider N executions of I, for N being large enough. Thus the distribution *DSys* will have a mean value of $N.T$ while that of *DStealth* will be of $N.(T + \delta)$. The value $N.\delta$ for suitable N will be large enough to provide a significant decision rule.

5 Conclusion and Future Works

In this paper, we have proposed an efficient information theoretic-based formal model for stealth program based on similarities with steganographic techniques. Our formalisation enables to exhaustively identify all the possible stealth techniques. At the present time, all the known stealth mechanism used by detected malware belong to the most simpliest classes.

We have also derived a statictical model for stealth detection and the main conclusion is that unconditionnally secure stealth is likely to exist only with respect to given detection estimators, when considering testing simulability.

Whether unconditionnally secure stealth whatever may be the detection estimators considered is still an open problem.

Stealth is often considered as a malevolent techniques. However, it could be used for beneficial applications. In an opposite view, stealth could provide a powerful security environment for computer sustem to hide active security mechanisms to the attackers. Current research in connection with the DCNS aims at using stealth technologies to embed counter-measure-like protection of critical computer systems in Navy Combat environments in order to prevent computer attacks: the attacker must not detect the presence of security element and thus stealth technologies precisely aims at that purpose.

References

1. Cachin C. (1998), An Information-theoretic Model for Steganography, *Information and Computation*, vol. 192, pp. 41–56.
2. Cachin C. (2005), Digital Steganography. In: *Encyclopedia of Cryptography and Security*, Springer Verlag.
3. Cohen F. (1986), *Computer viruses*, Ph. D thesis, University of Southern California, January 1986.
4. Filiol E. (2007), *Techniques virales avancées*, Collection IRIS, Springer Verlag France. An English translation is pending and will be in print in Fall 2007.
5. Filiol E. and Josse, S. (2007), A Statistical Model for Viral Detection Undecidability. *Journal in Computer Virology*, EICAR'07 Special Issue, V. Broucek ed., 3 (2).
6. Filiol E. (2007), Formalisation of Robustness and Capacity of Stealth Techniques. Technical Report 2007-V8.
7. Goldberg R. P. (1974), Survey of Virtual Machine Research. *IEEE Computer*, 7, pp. 34–45.
8. Hoglund G. et Butler J. (2006), *Rootkits: Subverting the Windows Kernel*, Addison Wesley, ISBN 0-321-29431-9.
9. Hopper N. J., Langford J. and van Ahn (2002), Provably Secure Steganography. In: *Advances in Cryptology - CRYPTO'02*, LNCS 2442, pp. 77-92, Springer.
10. Josse S. (2007), Rootkit Detection from Outside the Matrix. *Journal in Computer Virology*, EICAR'07 Special Issue, V. Broucek ed., 3 (2).
11. Kerckhoffs A. (1883), *La cryptologie militaire*, Louis BAUDOIN éditeur, Paris.
12. Lauradoux C. (2007), *Conception et synthèse en cryptologie symétrique.* Thèse de doctorat (PhD Thesis), Ecole Polytechnique, Palaiseau, juin 2007.
13. Lauradoux C. (2007), Detecting Virtual Rootkits with Covert Channel. *Journal in Computer Virology*, to appear.
14. King S. T., Chen P. M., Wang Y.-M., Verbowski C., Wang H. J. et Lorch (2006), SubVirt: Implementing Malware With Virtual Machines, Université du Michigan et Microsoft Research, `http://www.eecs.umich.edu/Rio/papers/king06.pdf`
15. Lehman E. L. (1959), *Testing Statistical Hypotheses*, Wiley, New York.
16. Popek G. J. and Goldberg R. P. (1974), Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17 (7), pp. 412–421.
17. Robin J. S. and Irvine C. E. (2000), Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. *Proceedings of the 9th USENIX Security Symposium.*

18. Rutkowska, J. (2005), System Virginity Verifier, Defining the Roadmap for Malware Detection on Windows System. *Hack In The Box Security Conference*, September 28-29th, Kuala Lumpur, Malaysia.
19. Rutkowska J. (2006), Subverting Vista Kernel for Fun and Profit, *SysCan'06 Conference*, July 21st, Singapore. Available at `http://invisiblethings.org/papers`
20. Rutkowska J. (2006), Introducing Stealth Malware Taxonomy, Version 1.01, `http://invisiblethings.org/papers`.
21. Rutkowski J. (2002), Execution Path Analysis: Finding Kernel Based Rootkits. `http://www.phrack.org/archives/59/p59-0x13`
22. Zuo Z. and Zhou M. (2004), Some further theoretical results about computer viruses, *The Computer Journal*, Vol. 47, No. 6.