J Comput Virol (2007) 3:23–37
DOI 10.1007/s11416-006-0026-9

EXTENDED VERSION OF WTCV'06

Evaluation methodology and theoretical model for antiviral behavioural detection strategies

Eric Filiol · Grégoire Jacob · Mickaël Le Liard

Received: 29 July 2006 / Revised 28 August 2006 / Accepted: 15 September 2006 / Published online: 19 October 2006

Abstract Behavioural analysis for detection of malware has recently emerged as a new promising set of antiviral techniques: function-based detection is now considered along with sequence-based detection. Most of the antivirus publishers now claim to use behavioral analysis as a marketing argument. But the real impact of these "new" techniques seems to be mitigated since no real progress in the general antiviral fight has been noticed nowadays. This paper presents an evaluation methodology of the real capabilities of antivirus software with respect to the behavioral analysis. It is shown that contrary to the claims of some publishers, behavioural analysis is still very marginally used and implemented. These techniques are quite always either validated by or dependant on classical form-based detection methods (detection pattern as an example). In this context, we propose a generalised, theoretical detection model which considers at the same time both form-based and function-based detection and give some essential properties this model should exibhit to achieve a real behavioural-based detection.

This paper is the extended version of the paper presented at WTCV'06 (Workshop in Theoretical Computer Virology) in Nancy, France, May 2006.

E. Filiol (✉) · G. Jacob · M. L. Liard
Ecole Supérieure et d'Application des Transmissions,
Laboratoire de virologie et de cryptologie,
BP 18, 35998 Rennes, France
e-mail: efiliol@esat.terre.defense.gouv.fr

M. L. Liard
EQUANT, rue de la Touche Lambert,
35512 Cesson Sévigné, France
e-mail: mickael.leliard@orange-ft.com

Keywords Malware · Antivirus · Black-box analysis · Malware detection · Polymorphism · Metamorphism · Detection model

1 Introduction

Up to this point, the introduced evaluation methods mainly audit the effectiveness of form-based detection engines [7,10]. However, these engines experience important and evergrowing difficulties to deal with the development of polymorphic/metamorphic techniques. Behavioral blockers and more globally every function-based detection method prove particularly adapted to address anti-antiviral techniques such as polymorphism/metamorphism.

The concept of behaviour-based detection has been originally introduced by Fred Cohen [3, p. 73]. Unfortunately, this kind of detection has been proved to be indecidable as its sequence-based counterpart is. However, behaviour-based detection appears to be a promising approach, even if there is a lot of technical problems to differentiate legitimate behaviours from purely malicious ones. Most of the reknowned antivirus publishers have decided to include it in their product. Behaviour-based detection has now become a marketing argument. What is the reality?

The issue to determine whether behaviour-based detection is indeed implemented is essential. Unless using disassembly techniques which are illegal in most countries, there is no other evaluation approach than black-box analysis. The most simple one consists in testing some reference codes and to check what the detection results really are. The most sophisticated ones use learning algorithms. But the most difficult problem is to be able to distinguish between sequence-based techniques and behaviour-based techniques. In most cases, there is no simple facilities supplied such as a case to tick in order to disactivate sequence-based detection.

The objective of this paper is twofold. Firstly we present an evaluation methodology of behaviour-based detection techniques. For this purpose, we consider behavioural polymorphism/metamorphism. In other words, instead of making the code sequence change like in classical techniques of polymorphism/metamorphism – which proves to be efficient to bypass sequence-based detection techniques [7] – we make the functions of the malware themselves change while the final malware actions remain the same. Thus it has been possible for the cases which have been considered to precisely identify what is really involved in the detection process.

Secondly, we extend the notion of detection scheme, which has been proposed in [7], by considering the generalised concept of detection strategy, which no longer distinguishes sequence-based detection techniques from behaviour-based techniques. A mathematical analysis of different detection strategies is given. The properties of the underlying detection functions are addressed as well.

This paper is organised as follows. In Sect. 2, we recall the theoretical notation and concepts which are used throughout the paper. In Sect. 3, we then present the behaviour-based detection evaluation method. In addition, in Appendix 5, we give detailed evaluation results for the main antiviruses which claim to use behavioural detection. Section 4 provides a mathematical analysis of detection strategies. In particular, some properties that good detection functions should present are detailed. At last, we will conclude and address some open-problems with respect to our work.

2 Generalised mathematical model for malware detection

In [7], a theoretical model has been proposed for sequence-based detection techniques.[1] The concept of *detection scheme* has been introduced. We recall hereafter its definition we will start from in the present paper.

Definition 1 *A malware* detection scheme *with respect to a given malware $\mathcal{M}$ is the pair $\{\mathcal{S}_{\mathcal{M}}, f_{\mathcal{M}}\}$. If we consider sequence-based detection the set $\mathcal{S}_{\mathcal{M}}$ will contain bytes whereas if behaviour-based detection is considered instead, this set contains program functions.*

[1] The present paper is based on notation developped in [7]. The reader is advised to read it before. Nonetheless, we recall here most of its basic notation and concepts for self-containment purposes.

Since this definition does not consider both sequence-based detection and behaviour-based detection at the same time, we propose the next definition as a generalisation of the notion of detection: we will now consider the more universal concept of *detection strategy*.

Definition 2 (Detection strategy) *A malware* detection strategy *$\mathcal{DS}$ with respect to a given malware $\mathcal{M}$ is the triplet $\mathcal{DS} = \{\mathcal{S}_{\mathcal{M}}, \mathcal{B}_{\mathcal{M}}, f_{\mathcal{M}}\}$, where $\mathcal{S}_{\mathcal{M}}$ is a set of bytes, $\mathcal{B}_{\mathcal{M}}$ is a set of program functions (behaviours) and $f_{\mathcal{M}} : \mathbb{F}_2^{|\mathcal{S}_{\mathcal{M}}|} \times \mathbb{F}_2^{|\mathcal{B}_{\mathcal{M}}|} \rightarrow \mathbb{F}_2$ is a Boolean detection function, $\mathbb{F}_2$ being the binary field.*

It is interesting to notice that this definition refers to both known and unknown malware. As far as an unknown code $\mathcal{M}$ is involved, the set $\mathcal{B}_{\mathcal{M}}$ is precisely the set which triggers an alert with respect to $\mathcal{M}$. If the essential nature of the set $\mathcal{S}_{\mathcal{M}}$ is quite easy to understand – it contains bytes – that of the set $\mathcal{B}_{\mathcal{M}}$ is probably not. In fact, this set can be considered as a meta-set of bytes, in the following sense: behaviours can be described by structures of bytes which more or less correspond to each procedure achieving a given action or behaviour. Reading a file or creating a mutex can be described by means of such more or less complex structures of bytes that can be located either on the hard disk (inactive malware code) or in memory (malware is active). From a formal point of view, thus $\mathcal{B}_{\mathcal{M}} \subset \mathbb{N}_{256}^{\infty}$ (a family of sequences of bytes of indefinite length). In the rest of the paper, we will only speak of behaviour. Asserting that behaviour $b \in \mathcal{B}_{\mathcal{M}}$ is realised means that the code contains a structure of bytes whose execution results in the behaviour b.

In order to consider Boolean functions thoroughly as detection functions, let us explicit this definition from a mathematical point of view. For that purpose, we adopt the formalism used in [7]. Let us describe the action of a given malware detector $\mathcal{D}$ on a file $\mathcal{F}$ that is suspected to be infected by $\mathcal{M}$. Let us first define the $s + b$ binary variables X_j $(1 \leq j \leq s + b)$ as follows:

$$X_j = \begin{cases} 1 & \text{if } \mathcal{F}(i_j) = b_{\sigma(j)}, \\ 0 & \text{otherwise.} \end{cases}$$

The purpose of these Boolean variables is to express the fact that a given byte at a given location, or a given behaviour (in fact a structure of bytes, see before) is realised ($X = 1$) or not present ($X = 0$) in the malware, no matter how in practice the antivirus is checking it. Thus, according to these notation, we indifferently consider sequence-based objects or behaviours. This notation enables to clearly describe the modification of $\mathcal{S}_{\mathcal{M}}$ bytes or $\mathcal{B}_{\mathcal{M}}$ behaviours by anyone who tries to bypass a given detector $\mathcal{D}$. Thus X_j equals 0 means that we have indeed modified $\mathcal{S}_{\mathcal{M}}(j)$ or $\mathcal{B}_{\mathcal{M}}(j)$ (up to the permutation σ). The reader will note that by associating any byte in $\mathcal{S}_{\mathcal{M}}$ or any behaviour in $\mathcal{B}_{\mathcal{M}}$ to Boolean variables, we thus can consider the boolean sets $\mathbb{F}_2^{|\mathcal{S}_{\mathcal{M}}|}$ and $\mathbb{F}_2^{|\mathcal{B}_{\mathcal{M}}|}$ respectively. We consequently have $s = |\mathcal{S}_{\mathcal{M}}|$ and $b = |\mathcal{B}_{\mathcal{M}}|$. For sake of simplicity, we will now consider, up to an isomorphism, only the Boolean set $\mathbb{F}_2^{|\mathcal{S}_{\mathcal{M}} \cup \mathcal{B}_{\mathcal{M}}|}$ which has a cardinal of $2^n = 2^{s+b}$.

Let us note that σ [7] denotes a bijective permutation over the byte of $\mathcal{S}_{\mathcal{M}}$. The use of this permutation enables to take into account potential modification of $\mathcal{M}$ made by any copycat. Indeed, different code obfuscation or polymorphism/metamorphism techniques can modify the structure (byte ordering or indexing) of $\mathcal{S}_{\mathcal{M}}$ or the sequence of behaviours of $\mathcal{B}_{\mathcal{M}}$.

Let us now consider a Boolean function $f_{\mathcal{M}} : \mathbb{F}_2^n \to \mathbb{F}_2$. We say that a given detector decides that $\mathcal{F}$ is infected by the malware $\mathcal{M}$ with respect to the *detection function* $f_{\mathcal{M}}$ and the malware detection components $\mathcal{S}_{\mathcal{M}} \cup \mathcal{B}_{\mathcal{M}}$ is and only if $f_{\mathcal{M}}(X_1, X_2, \ldots, X_n) = 1$. In other words:

$$f_{\mathcal{M}}(X_1, X_2, \ldots, X_n) = \begin{cases} 1 & \mathcal{F} \text{ is infected by } \mathcal{M}, \\ 0 & \mathcal{F} \text{ is not infected by } \mathcal{M}. \end{cases}$$

Here the function models the different possible combinations of bytes or behaviours that result in effectively detecting the malware. 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.

Analyzing antivirus software aims at identifying which characteristics are considered by the software: behaviours, detection patterns, mode of detection…As previously shown in [7], such an approach consists in solving the characteristic extraction problem which has been shown equivalent to a learning problem. For that purpose, the analyst may selectively modify any detected malware in order to decide whether the modification results in detection or not. With the previous formalism, this problem comes to computing the *non-detection* function $\overline{f_{\mathcal{M}}}$. In other words $\overline{f_{\mathcal{M}}} = 1 \oplus f_{\mathcal{M}}$. This function describes the way malware may be modified to bypass the detection strategy $\{\mathcal{S}_{\mathcal{M}}, \mathcal{B}_{\mathcal{M}}, f_{\mathcal{M}}\}$ in force. These modifications correspond to the n-tuples $(x_1, x_2, \ldots, x_n)$ for which the non-detection equals 1. For a given such n-tuple, any modification that can be applied is defined as follows:

$$\begin{cases} \text{if } x_i = 0 & \text{byte or behaviour } i \text{ in } \mathcal{S}_{\mathcal{M}} \cup \mathcal{B}_{\mathcal{M}} \\ & \quad \text{must be modified,} \\ \text{if } x_i = 1 & \text{byte or behaviour } i \text{ in } \mathcal{S}_{\mathcal{M}} \cup \mathcal{B}_{\mathcal{M}} \\ & \quad \text{may be left unmodified.} \end{cases}$$

In the rest of the paper, we will indifferently consider either the detection function or the non-detection function. We will just give the following definition that will be later of some interest.

Definition 3 *A malware* bypassing strategy $\mathcal{BS}$ *with respect to a given malware* $\mathcal{M}$ *is the triplet* $\mathcal{BS} = \{\mathcal{S}_{\mathcal{M}}, \mathcal{B}_{\mathcal{M}}, \overline{f_{\mathcal{M}}}\}$.

Remark The malware $\mathcal{M}$ is uniquely characterised by both $\mathcal{S}_{\mathcal{M}}$ and $\mathcal{B}_{\mathcal{M}}$ as well as the detection function $f_{\mathcal{M}}$. This function allows to greatly reduce the false positive rate, when judiciously selected [7].

3 Behaviour-based detection evaluation method

The goal of this study has been to evaluate the effectiveness of behaviour-based detection in modern antivirus products. Our intention is not to uselessly criticise one or more products but to pinpoint the existing weaknesses in such detection techniques. We intend to propose an efficient evaluation methodology that should help anyone who is charge of antivirus evaluation. Despite the fact that our approach and results have considered a single malware – the *W32.MyDoom* mass-mailing worm – this can be generalised to any other malware.

In this study, we analysed the fact that modern malware detection techniques are supposed[2] to consider both detection pattern databases and forbidden behaviours databases. Then, we generalised the approach developped in [7] which consists in solving the problem of detection techniques extraction. In other words, any analyst aims at precisely guessing how any antivirus is working by performing a black-box analysis. In the present case, we selectively modify detection behaviours instead of detection patterns and record whether the tested antivirus still detect the modified code or not. The mathematical analysis of the results then enables to reconstruct the whole detection strategy as defined in Sect. 2.

Following this principle, we have chosen to assess the resistance of antiviral products with respect to behavioural detection, by examining their reactions face to different versions of a virus whose one or more functions have been modified. In other words, our approach

[2] This assertion has been deduced from the different unequivocal marketing claims of most of the antivirus publishers whose products we have tested in this study.

conceptually consists in realizing behavioural polymorphism/metamorphism instead of classical (form-based) one. Let us now present how we realised this new kind of code polymorphism/metamorphism on the *W32.MyDoom* mass-mailing worm.

3.1 *W32.MyDoom* behavioural polymorphism/metamorphism

The underlying idea is in fact to simulate the polymorphic/metamorphic generation of the virus code with respect to its functionnalities but in a selective and controlled way. To achieve this, we have generated separate different versions of a virus that could be the result of several duplications from an original strain. For each newly generated version, one of the main infectious behaviours has been manually modified, replaced or even suppressed simulating the different functional mutations. Unfortunately, regarding the state-of-the-art, these modifications are still hardly automatically performed. The concept of functional or behavioural polymorphism/metamorphism has still to be studied in-depth.

Throughout this article, we will focus our approach on the well-known virus *W32.MyDoom* [4,6,8,9]. The reason for this choice is quite simple. This mass-mailing worm may be considered as a reference declined in a whole range of versions and whose techniques keep being reused. It is clearly possible to reproduce and adapt the process to other viruses as far as they offer a minimal diversity of behaviours.

3.1.1 Identification of the behaviors

Before proceeding with any behaviour modification, the first step was to identify the main classes of infectious behaviors implemented in the malware. Considering behaviours is far more complex that considering detection patterns or other form-based object like in [7]. It was thus necessary to adopt a different approach and consider the source code[3] of the malware.

The tight functionnal analysis of the *W32.MyDoom* source code has made it possible to identify the flow of the different functions and their internal process. Each infectious behaviour manifests itself as one or several characteristic actions of the virus. An action can be either a particular functionality or simply a programming technique used during the virus development. Table 1 lists the main behaviours identified in *W32.MyDoom* as well as examples of the actions that have given away their presence.

[3] This source code may be either public or obtained through a disassembly step.

Table 1 Identified behaviors in MyDoom

Reference	Behavior	Actions
DUPLI	Code replication	Write the running file in the system directory
RESID	Residency	Virus is made memory resident by means of a run register key
PROPA	Spread	Massmailing with the virus as an attachment
OVINF	Overinfection test	Test whether a given a register key does exist
ACTIV	Activity test	Test whether a Mutex object is active in memory
STEAL	Stealth	Sets its own network protocol stack
POLYM	Polymorphism	Encryption of the embedded backdoor library
INFOR	Information gathering	Recursive scanning of user's and Internet files
FINAL	Final Payload	Backdoor library installation
SOCIA	Social engineering	Simulating lost e-mails recovery

Behaviours listed in Table 1 thus represents the actual set $\mathcal{B}_{\mathcal{M}}$ that has been presented in Sect. 2. We will note here $\mathcal{B}_{\mathcal{M}} = \{b_1, b_2, \ldots, b_{10}\}$ where b_i is one of the behaviour listed in Table 1.

3.1.2 Code modification with alternative behaviors

Once the behaviors which may be or are likely to be involved in the detection have been identified, the second step consists in modifying them according to the principle "different action/same result". In other words, the modification aims at performing polymophism/metamorphism at the functional (behavioural) level and not at the form level (the latter corresponding to the classical polymophism/metamorphism techniques). According to the notation of Sect. 2, we consider the ten Boolean variables $X_1, X_2, \ldots, X_{10}$ with respect to $\mathcal{B}_{\mathcal{M}} = \{b_1, b_2, \ldots, b_{10}\}$. Each time original behaviour b_i is left unmodified we have $X_i = 1$ whereas $X_i = 0$ each time the corresponding behaviour has been modified. The final aim consist thus to determine whether behavioural detection is really used and if any, which behaviours are involved.

This step is essential since the amount of information gathered during the analysis step proved to be proportional to the relevance of the newly generated versions. It is important to recall that we have to think in term of program functionalities to proceed. In other words we have to find equivalent potential actions (behaviour) that eventually lead to the same resulting malware action than the original strain. Table 2 summarises the alternative actions we have worked out for the

Table 2 Nature of the modifications brought to the behaviors

Behavior	Reference	Nature of the modifications
DUPLI	DUP_SH_CUT	Recopy in a shortcut file
	DUP_NAM_PATH	Recopy under a compressed system patch directory
RESID	RES_SERV_KEY	Virus inscription under a service register key
	RES_WIN_INI	Modification of the win.ini file
OVINF	INF_DIF_KEY	Existence test of a different register key
	INF_SUP_HID	Existence test of a “superhidden” file
	INF_ENV_VAR	Existence test of an environment variable
ACTIV	ACT_MUTEX	Existence test of a different mutex
	ACT_EVENT	Existence test of an event object
POLYM	POL_PLAIN_LIB	Backdoor library integrated as plaintext
	POL_FLOW_LIB	Backdoor library encrypted a stream cipher
	POL_PLAIN_STR	Strings left as plaintext
	POL_FLOW_STR	Strings flow encrypted
FINAL	FIN_TRIG_TARG	Different target and trigger for the DDOS attack
	FIN_NO_BDOOR	Suppression of the backdoor library

different behaviours considered for detection by the antivirus we have tested. Additional modifications could have clearly been considered but they prove to be unnecessary regarding the scope of the current study. Indeed every tested antivirus proved to implement behavioural detection at a very poor or limited level (see Appendix 5). Simple modifications manage to bypass them easily.

As an example, let us detail two of the modifications that are listed in Table 2. A complete description of the modifications we have performed are described in [9, pp. 26–29].

Overinfection test. The aim for the malware is to test whether it has previously infected the current host. In the *W32.MyDoom* case, this is checked by looking for a particular key (the infection marker) in the Windows base registry. A first variant would consist in modifying the infection marker itself but this would correspond to classical polymorphism (form-based).

Among many other possibilities, we choose to test whether a superhidden file is present (host infected) or not (host non-infected). This type of file makes it possible to hide any file even when the display of hidden files/directories is activated. In order to force superhidden files to be visible, administrator permissions as well as modification of base registry keys are required. The reader will note that this feature is undocumented in the Windows Operating System documentation. To use this file type, we have just to define the following parameter:

```
#define FILE_ATTRIBUTE_SUPER-HIDDEN
         0x00000006
```

To mark the infection, the virus uses the *CreateFile* function to create a superhidden file in a secret location. To check for previous infection, the virus looks for this file.

Activity test. In this case, the *W32.MyDoom* code looks for a particular *Mutex*. The first obvious modification suggests to change the mutex’name but once again this does not appear conceptually very different from classical polymorphic techniques (form-based modifications). Many other Windows objects enables the interprocess synchronization. Therefore we considered a second variant in which the activity is tested by means of a different event object. Yet simple, we used the following one:

```
TaskmonInitialised     Event
```

In this case – as in many other cases – the code checks the function error exit code whenever the object is created.

3.2 Test methodology

In order to apprehend the test procedure, it is important to keep in mind that most of the actual antiviral products do not provide a clear distinction between sequence-based detection and behaviour-based detection. It is almost always impossible to determine which method is really used. From a practical point of view, very few controls – if any – are left to the user, in particular to activate separately the different detection methods, which would have proved helpful for the test. As a general rule, we can assert that any “exact” identification is based on sequence-based detection while “generic” identification may or might involve behaviour analysis at some level.

Since it is impossible to run the behavioral analysis without running the form-based analysis as noticed in our experiments, it becomes essential to extract the sequence-based detection components (e.g. signature) first. Thus, once this signature is removed, the virus detection can eventually rely on function-based methods only such as behavioural engines. As a consequence the test procedure has been broken down into two phases. During the first one, we have tested the different versions statically by manual scanning in order to identify the nature and the location of the sequence-based components. For the second phase, we have activated the resident protection which is supposed to run the behavioural analysis.[4] Thanks to the knowledge about the signature extracted in the first place, it becomes now possible to retrieve information on the true nature of the behavioural detection.

[4] We did not address the case of code emulation. The reason lies in the fact that existing products only consider sequence-based detection techniques for detection, especially when classical (form-based) polymorphism/encryption is involved. A future evolution of antivirus could be to implement behaviour-based detection during code emulation steps as well.

Let us now describe our test bench. The experiments require several Windows 2000 platforms which are clean from malware and deconnected from any network. A recovery solution must be considered in order to reinitiate the platforms in their original state after an infection. To achieve this, virtually emulated machines such as *VMware* prove to be particularly adapted. Indeed the execution remains confined to a sandbox and by simply reloading the original configuration, it resets the platform. A second, less convenient solution is to reboot after each test on a clean ghost of the operating system.

The detailed results are given product by product in Appendix 5.

4 Mathematical analysis of detection strategy

4.1 Global synthesis of the experiments results

To reach a higher level of abstraction, we have now to detach from the products in order to introduce a global trend according to the formalization we have established in the first place.

It must be clear that all the realised deductions mainly concern the detection strategies with respect to a given malware. No particular information has been gathered about the different implementations of the behavioural detection really adopted by the editors, if any. It simply seems that in most cases, the modelization by suspect scenarios has been chosen by the antivirus developpers. In fact, this detection mode proves specifically adapted to this model given the fact that only the dangerous behaviours have been identified and modified. We could not have obtained more relevant results if antiviral products had chosen a modelization by legitimate behaviours. This last aspect relates more to IDS concepts [12].

As far as detection strategy is concerned, on the other hand, a very strong trend comes out with regards to the results that no antiviral product really contradicts except Viguard. This trend can be formulated through the two main following hypotheses:

- $\mathcal{H}_1$: behavioural detection is non implemented or inefficient,
- $\mathcal{H}_2$: behavioural detection is ignored without corroboration or validation by a sequence-based detection component (e.g a signature).

For Viguard, we can establish an additional hypothesis aside:

- $\mathcal{H}_3$: behavioural detection consists in detecting any possibly threatening behaviour.

In the scope of our tests, a simple and significant fact backs up these two first hypotheses. Whenever a virus has been detected, a precise response has been provided including its identity. This can be achieved only through a signature; a pure behavioural detection would have detected a generic agent. From a more mathematical point of view, using a Boolean modelisation as introduced in Sect. 2, the three hypotheses can be translated as follows:

$$\mathcal{H}_1 : T_{\text{sig}}$$
$$\mathcal{H}_2 : T_{\text{sig}} \vee (T_{\text{sig}} \wedge T_{\text{behav}}) = T_{\text{sig}} \quad \text{(by absorption law)}$$
$$\mathcal{H}_3 : T_{\text{behav}} = \bigvee_{i=0}^{b} X_i \quad \text{where } X_i \text{ relates to the } i\text{th element in the behaviour base } \mathcal{B}_{\mathcal{M}}$$

The notation T_{sig} relates to the restriction $f_{\mathcal{M}}^{\mathcal{S}_{\mathcal{M}}}$ of the detection function $f_{\mathcal{M}}$ to the set $\mathcal{S}_{\mathcal{M}}$. This means that Boolean variables with respect to the set $\mathcal{B}_{\mathcal{M}}$ do not appear in the detection (sub-)function disjunctive normal form. Conversely, as far as the $\mathcal{H}_3$ hypothesis is concerned, we consider the restriction $f_{\mathcal{M}}^{\mathcal{B}_{\mathcal{M}}}$ to the set of behaviours $\mathcal{B}_{\mathcal{M}}$ with the essential differences that:

- the set $\mathcal{B}_{\mathcal{M}}$ contains any behaviour that might be "dangerous",
- the detection function is the OR function, of weight $2^{|\mathcal{B}_{\mathcal{M}}|} - 1$.

By setting the problem mathematically, it becomes clear that the first two hypotheses are in fact equivalent and lead to a same and unique conclusion. In both cases, we can say that the utility of the behavioural engine, if any, is questionable. Without any doubt, it is due to the fact that it is not properly integrated into the detection strategy. On behalf of decreasing the rate of false positives, this trade-off solution totally inhibits the main functionality of the behavioural detection: the detection of unknown viruses. If we examine the two logical formulae, they remain really simple ones and represent a negligible part of the Boolean modelisation potential. Nothing proves that they necessarily are the most optimal. On the other hand, Viguard's hypothesis is exactly the opposite. The maximum weight is given to each behaviour detection, thus increasing alarmingly the rate of false positives.

It is getting even worse knowing that several antiviral products like AntiVirusKit, KAV and F-secure partly or totally share the same sequence-based detection features (e.g. signature databases). Considering the efficiency factor, they are completely equivalent and combining them would not increase the rate of detection.

Only the behavioural engine properly implemented would have made possible a real distinction.

4.2 Mathematical properties of efficient detection strategies

In [7], the properties, that any good detection scheme should exhibit, have been proposed as well as a secure detection scheme. All of them extend to the concept of detection strategy as defined in Sect. 2. We just have to consider two reference databases (bytes and behaviours) instead of one. The only difference lies in the detection function (or the non detection function on the attacker's side) since its domain is now twofold. From a general point of view, we claim that the input of these two functions – Boolean variables describing bytes and behaviours – must have an equal effect on their output. It is not the case in existing antivirus software.

Before giving some results for interesting detection functions, let us first make the theoretical background more precise.

4.2.1 Properties of Boolean detection functions

As previously exposed, the detection function plays an essential role in a detection strategy. It represents the way form-based and/or behaviour-based characteristics are analysed and searched for in a file. In [7], a strong result show that the weight of the detection function (the number of input for which the function outputs 1) was an important parameter. While a weight of a single unit (AND detection function) proves to be the poorest solution, detection functions with a larger weight offer a wider range of detection opportunities (see [7] and wildcard detection as an example). As far as the detection function is concerned, this remark remains true – but in the attacker's opposite view – since we have for a Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2$:

$$\mathrm{wt}(f) = 2^n - \mathrm{wt}(\bar{f}),$$

where $\mathrm{wt}(f) = |\{x|f(x) = 1\}|$. In other words, if the detection function has a small weight (limited number of detection configurations), the non detection function will exhibit a larger weight (a high number of bypassing configurations). This enables to consider the following definition.

Definition 4 *Let be a detection strategy $\mathcal{DS} = \{\mathcal{S_M}, \mathcal{B_M}, f_{\mathcal{M}}\}$ and the corresponding bypassing strategy $\mathcal{BS} = \{\mathcal{S_M}, \mathcal{B_M}, \overline{f_{\mathcal{M}}}\}$ Let $n = |\mathcal{S_M}| + |\mathcal{B_M}| = s + b$ The strategy $\mathcal{DS}$ is said to be* stronger *than the strategy $\mathcal{BS}$ if and only if*

$$2^{n-1} \leq \mathrm{wt}(f_{\mathcal{M}}) \leq 2^n - 1$$

This first definition can be considered as a first criterion to select "good" detection functions.

Another very important criterion deals with the respective influence of Boolean variables X_i on the detection function's output. It is nothing but essential that they all have the same impact on $f_{\mathcal{M}}$ (or equivalently on $\overline{f_{\mathcal{M}}}$). If it was not the case – due to the fact that a given variable could have a preponderant role (respectively a marginal role) compared to the other input variables – the relevant byte/behaviour should be considered as preponderant (respectively marginal) in the detection strategy. Such an information would inevitably be exploited by any attacker in order to bypass the detection strategy considered. This can be generalised to any t-set of Boolean input variables. We will adopt the following definition.

Definition 5 *A detection function is said to be* weakly bypassable at order *t if and only if the function's output does not statistically depend on any set of input variables of size at most t. A detection function is said to be* strongly bypassable at order *t if and only if its outputs statistically depend identically on any set of at most t input variables.*

According to this definition, no particular set of at most t input variables will be more interesting to consider than another one, in order to bypass the detection strategy. The difference between "weakly" and "strongly" lies in the fact that in the first place there is no dependance with respect to any t-set of input variables whereas in the second case such a dependance does exist but with respect to any t-set, identically. It is rather intuitive to assert that realizing the first case is more difficult than the second one. We will demonstrate it in Sect. 4.2. Let us mention that if $f_{\mathcal{M}}$ is (weakly or strongly) bypassable at order t, this property still holds for the corresponding non-detection function $\overline{f_{\mathcal{M}}}$ – but the meaning in attacker's approach is reversed.

This property enables to logically consider a very particular class of Boolean functions, which are very important in symmetric cryptography: correlation immune functions. In order to consider these special functions, let us first recall the main mathematical tool which is used to characterise the concept of correlation for Boolean functions. The reader may refer to [1, Chap. 2] or [11, p. 207] for more details on this mathematical tool.

Definition 6 *Let f be a Boolean function over $\mathbb{F}_2^n$. The* Walsh-Hadamard transform *of f is the Fourier transform of the corresponding sign function, $x \mapsto (-1)^{f(x)}$:*

$$\forall\, u \in \mathbb{F}_2^n,\ \widehat{\chi_f}(u) = \sum_{x \in \mathbb{F}_2^n} (-1)^{f(x)}(-1)^{<u,x>}$$

where $< .,. >$ denotes the usual scalar product.

The Walsh-Hadamard transform enables to exhibit statistical dependencies (biases) between some subsets of input variables and the function's output. Let us make things more precise.

Definition 7 *A Boolean function in n variables is said to be correlation immune at order t if its statistical distribution of output does not change when at most t input variables have a fixed value.*

In other words, the function's ouptut is statistically independent of any variable vector $(X_{i_1}, X_{i_2}, \dots, X_{i_t})$. As an example, the function

$$f(X_1, X_2, \dots, X_n) = X_1 \oplus X_2 \oplus \dots X_n$$

is correlation immune at order $(n-1)$.

A very important result [15], establishes a link between correlation immunity and Walsh transform coefficients.

Proposition 1 [15] *The Boolean function f in n variables is correlation-immune at order t if and only if we have*[5]

$$\widehat{\chi_f(u)} = 0 \quad \forall\, u \in \mathbb{F}_2^n,\ 1 \leq wt(u) \leq t.$$

We now can establish the following result.

Proposition 2 *A Boolean function $f_{\mathcal{M}}$ is weakly bypassable at order t if and only if it is correlation-immune at order t. A Boolean function $f_{\mathcal{M}}$ is strongly bypassable at the order t if and only if*

$$\forall\, u \in \mathbb{F}_2^n \text{ such that } 1 \leq wt(u) \leq t \qquad \widehat{\chi_f(u)} \text{ is a constant.}$$

Proof The proof is obvious by definition of correlation immunity. Let us notice that this property still holds for non detection function $\overline{f_{\mathcal{M}}}$ since we have

$$\widehat{\chi_f(u)} = -\widehat{\chi_{\overline{f}}(u)}.$$

□

This proposition makes it possible to define a criterion to choose detection function for real behaviour-based detection. Such a function should be weakly bypassable (respectively strongly bypassable) at least with respect to variables related to $\mathcal{S}_{\mathcal{M}}$.

4.2.2 Some good detection functions

According to Definition 4, the weight of detection function must be at least equal to 2^{n-1}. But known results in cryptography show that balanced functions – functions of weight exactly equal to 2^{n-1} – are among interesting candidates as soon as correlation immunity is concerned. One of the basic reasons is that it is rather easy to find balanced functions having some desirable properties than overbalanced ones. This is why we will focus on balanced Boolean functions.

[5] wt(u) denotes the Hamming weight of u, that is to say the number of 1 in the binary expansion of u.

Weakly bypassable detection functions. Linear functions are obviously the first functions to consider. Their algebraic normal form is given by

$$f(X_1, X_2, \dots, X_n) = X_1 \oplus X_2 \oplus \dots \oplus X_n.$$

The most interesting property as far as detection functions are considered lies in the following proposition.

Proposition 3 *The function $X_1 \oplus X_2 \oplus \cdots X_n$ is weakly bypassable at order $n-1$.*

Proof By Walsh transform computation, we show that the only non-zero Walsh coefficient $\widehat{\chi_f(u)} = 2^n \neq 0$ is that for $u = (1, 1, 1, \dots, 1)$. Hence the result. □

This proposition means that unless considering simultaneously all the input variables – in other words all the bytes and all the behaviours, in the context of malware detection – no particular subset is worth considering in order to bypass detection when linear detection function are used.

Strongly bypassable detection functions Another very interesting detection functions are MAJORITY functions. Indeed, they are strongly bypassable.

Definition 8 *We call a n variable Boolean function a* MAJORITY *function (denoted* MAJ$_n$*, the Boolean function which maps $\mathbb{F}_2^n$ to $\mathbb{F}_2$ such that:*

$$f(x) = 1 \Leftrightarrow \begin{cases} wt(x) \geq \frac{n+1}{2} & \text{if } n \text{ impair,} \\ wt(x) \geq \frac{n}{2} + 1 & \text{if } n \text{ pair.} \end{cases}$$

Moreover, when n is even, $f(x) = 1$ for exactly $\frac{1}{2}\binom{n}{\frac{n}{2}}$ input values x of weight $\frac{n}{2}$.

While there are $\begin{pmatrix} \binom{n}{\frac{n}{2}} \\ \frac{1}{2}\binom{n}{\frac{n}{2}} \end{pmatrix}$ functions MAJ$_{2p}$, there is only one MAJ$_{2p+1}$ for a fixed value of p. Without loss of generality, we will consider only the last case ($n = 2p+1$). A known result [2] asserts that MAJ$_n$ functions (n even or odd) are balanced for any value of n.

The next proposition shows that MAJ$_n$ are strongly bypassable functions, except asymptotically.

Proposition 4 *The Boolean functions* MAJ$_n$ *are correlation-immune at order 0, with respect to every variables x_i and*

$$P[\text{MAJ}_n(x) = x_i] = \frac{1}{2} + \frac{\binom{n-1}{\frac{n-1}{2}}}{2^n}.$$

The reader will find the proof of this proposition in [5]. This result asserts that if MAJ$_n$ functions are statistically dependent on every of their input variables, they are also identically dependent on them. Consequently, no particular variable (a byte of a behaviour) is playing a more important role than another one.

However in the context of detection functions, MAJ_n have an additional interesting property as stated by the following proposition.

Proposition 5 *Let us consider a* MAJ_{2p+1} *for a given value p. Its DNF formula contains* $\binom{2p+1}{p+1}$ *logical terms. The corresponding function* $\overline{\mathrm{MAJ}_{2p+1}}$ *contains* $\binom{2p+1}{p+1}$ *terms, each of them containing* $p+1$ *variables in negated form* ($\overline{x_i}$).

Proof We will prove the second part of the proposition by induction on p. Let us consider a $\overline{\mathrm{MAJ}_{2p+1}}$ function. Let us denote H_{2p+1} the property we want to prove. Firstly, it is obvious that H_1 holds since $\overline{\mathrm{MAJ}_1}$ has $\overline{x_1}$ as DNF. Let us suppose that H_{2p+1} holds and let us show that consequently H_{2p+2} still holds.

Since H_{2p+1} is true, the corresponding DNF has $\binom{2p+1}{p+1} = \binom{2p+1}{p}$ terms. Let us build the DNF of function $\overline{\mathrm{MAJ}_{2p+3}}$ from that of $\overline{\mathrm{MAJ}_{2p+1}}$. For that purpose, we consider two additional Boolean variables $y_1 = x_{2p+2}$ and $y_0 = x_{2p+3}$. For sake of simplicity, let us note $\{x_p\}$ the logical union of terms in $\overline{\mathrm{MAJ}_{2p+1}}$ each of them containing exactly p negated variables. In the same way, we will note $\{x_{\leq p}\}$ the logical union of terms containing at most p negated variables.

The DNF formula of $\overline{\mathrm{MAJ}_{2p+3}}$ is given by

$$10\{x_p\} \vee 01\{x_p\} \vee 11\{x_{p-1}\} \vee 00\{x_{p+1}\}. \tag{1}$$

where the two first bits describe the y_1 and y_0 additional variables. Let us mention that any other logical terms, whose general DNF formula is

$$01x_{\{\leq p-2\}} \vee 10x_{\{\leq p-2\}} \vee 11x_{\{\leq p-2\}} \vee 00x_{\{\leq p-2\}},$$

has to not be considered since they each outputs 0 with regards to the $\overline{\mathrm{MAJ}_{2p+3}}$ function. Since H_{2p+1} holds and by construction of the $\overline{\mathrm{MAJ}_{2p+3}}$ DNF formula, the union of logical terms in Eq. (1) cannot be simplified further. Let us now determine how many terms are involved.

Obviously, Eq. (1) includes a number of logical terms which is given by

$$\binom{2p+1}{p} + \binom{2p+1}{p} + \binom{2p+1}{p+1} + \binom{2p+1}{p-1}$$
$$= \frac{2(2p+3)}{(p+2)}\binom{2p+1}{p} = \binom{2p+3}{p+1}.$$

Hence the result for the number of logical terms in DNF formula. By construction, it is easy to check that each logical term contains indeed $p+1$ negated variables.

The first part of the proposition is proved by induction in the same way. □

Remark The number of logical terms in the DNF formulae (with regards to both MAJ_{2p+1} and $\overline{\mathrm{MAJ}_{2p+1}}$) corresponds in fact the maximum size of an antichain in a poset (Sperner's theorem [14]). By considering the set $\mathcal{F}$ of all p-element subsets of the Boolean set $\mathbb{F}_2^{2p+1}$ (each of them corresponds to the support of a DNF logical term), we obtain the result since $\mathcal{F}$ is an antichain. Therefore, by basic properties on antichains, we cannot simplify the corresponding DNF further.

This results asserts that if we choose a MAJ_{2p+1} as a detection function, thus any attacker who will to bypass a detection strategy, will have to modify at least $p+1$ variables (in other words bytes or behaviours). Consequently, by choosing large value of p, the attacker will face a high complexity [7].

Example 1 Let us consider the MAJ_5 function whose DNF formula is

$$\begin{aligned}\mathrm{MAJ}_5 = {} & x_4x_3x_2 \vee x_4x_3\overline{x_2}x_1 \vee \overline{x_4}x_3x_2x_1 \vee x_4\overline{x_3}x_2x_1 \\ & \vee x_4x_3\overline{x_2x_1} \vee x_4\overline{x_3}x_2\overline{x_1}x_0 \vee x_4\overline{x_3x_2}x_1x_0 \\ & \vee \overline{x_4}x_3x_2\overline{x_1}x_0 \vee \overline{x_4}x_3\overline{x_2}x_1x_0 \vee \overline{x_4x_3}x_2x_1x_0\end{aligned}$$

The DNF formula of the corresponding non detection function is then:

$$\begin{aligned}\overline{\mathrm{MAJ}_5} = {} & \overline{x_0x_1x_2} \vee \overline{x_0x_1}x_2\overline{x_3} \vee \overline{x_0x_1}x_2x_3\overline{x_4} \vee \overline{x_0}x_1\overline{x_2x_3} \\ & \vee \overline{x_0}x_1\overline{x_2}x_3\overline{x_4} \vee \overline{x_0}x_1x_2\overline{x_3x_4} \vee x_0\overline{x_1x_2x_3} \\ & \vee x_0\overline{x_1x_2}x_3\overline{x_4} \vee x_0\overline{x_1}x_2\overline{x_3x_4} \vee x_0x_1\overline{x_2x_3x_4}\end{aligned}$$

The non-detection function will equal 1 if and only if at least three detection features (bytes in the code or behaviours) are simultaneously modified.

We could consider a special class of strongly bypassable functions: *bent* functions [13]. These functions f in n variables verify $|\widehat{\chi_f}(u)| = 2^{\frac{n}{2}}$ for all $u \in \mathbb{F}_2^n$. Unfortunately, exhibiting bent functions is still an open problem unless for some trivial or simple class and for small values of n.

5 Conclusion and future work

In this paper, we have proposed a generalised model for malware detection which considers at the same time both sequence-based detection and behaviour-based detection. As far as the latter is concerned, we have proposed an evaluation methodology for behaviour engines of existing products. This partial study has shown that behaviour-based detection seems to be more a claim than a reality – at least for the antivirus we have tested.

This study has to be widely extended to other kinds of malware to produce a huge number of results that could produce an in-depth evaluation. By the present time, the methodology's validity itself has only been successfully tested. Future work will consider automated tools to produce behavioural polymorphism/metamorphism directly on input malware. We thus will be able to extensively evaluate behaviour-based detection in commercial products.

The secure detection scheme presented in [7] can be extended without any difficulty from detection schemes to detection strategies. The main interest lies in the fact that many malware source codes are publicly available. Consequently, an attacker will try to modify them rather at the functional level than at the code sequence level.

The work presented in this paper shows that up to now behaviour-based detection is not really implemented in existing antivirus software. The lack of a thorough model for such a detection seems to be the reason why it is not efficiently implemented yet. Our work may be a first step to precisely define what behavioural detection really is and how antivirus products should implement it. But there is still a growing need to study the concept of program behaviour from a theoretical point of view.

At last, this work should promote theoretical research on Boolean functions. Finding good detection functions, which both lower the number of false alarms (as pointed out in [7]) and make black-box analysis of antivirus software far more difficult at the same time, is an essential point in future developments of far more efficient antivirus products.

Acknowledgements The authors would like to thank reviewers for their valuable help in improving this paper as well as the TCV Special Issue editors Guillaume Bonfante and Jean-Yves Marion for their constant support in writing this paper.

Appendix A Detailed experimental results and interpretations

Seven antivirus softwares have been tested (Table 3). We have considered those who are supposed or claim to use behavioural detection. Fourth column gives the number of bytes involved in the detection pattern that has been extracted according to the techniques presented in [7]. The detection function is the AND logical function. Except for Avast, all detection patterns share the same following sub-pattern, whose bytes are located at indices:

$$1{,}080 \to 1{,}083,\ 1{,}090 \to 1{,}093,\ 1{,}100 \to 1{,}103,\ 1{,}111 \to 1{,}114.$$

We will now present the results obtained by confronting the test platforms to the new muted versions of the virus. Enough details have normally been provided to make sure that these results are reproducible, at least conceptually. Results on their own are of little interest, thus we will seize the opportunity to introduce the worked out reflections with regards to our problematic of the antiviral strategy evaluation methodology. We have willingly associated both to illustrate our speech and give quick references. This exercise of interpretation has been made product by product. Global synthetic results of the state-of-the-art of actual behavioural detection has been presented in Sect. 4.

A.1 Results product by product

Whatever may be the antivirus software, the summary of its results are stored in tables identically structured. Before anything else, it is important to briefly explain how to read each of the table entries. Each of them is associated to a particular modification labeled as in the reference Table 2. To support our argumentation we will punctually remind these references as elements of proof. Concerning the codification, an empty field means that no detection occured at all. A red one brings into light differences from the reference tests where the original executable and its included backdoor library have been separately tested.

AVG test results As a first comment, by consulting the content of the signature base, the antiviral product claims that six specific versions of *W32.MyDoom* are classified, plus a generic one. Though, with regards to the reference tests, the original strain is not detected by

Table 3 Tested antivirus software (versions and viral definitions)

Products	Version	Viral definition	Detection pattern size (in bytes)
Avast	4.6.763	0611-2	8
AVG	7.1.375	267.9.2/52	18,497
DrWeb	4.33.2.12231	10062006	637
F-Secure 2005	6.12-90	2006-06-02-02	46
G-Data	AVK 16.0.3	KAV-6.818/BD-16.864	41
KAV Pro	6.0	07062006	46
Viguard	11	NA	N/A

a static analysis whereas the backdoor library is detected as a generic version, once it has been both extracted and decrypted. From this established fact, two deductions can possibly be raised. Firstly, the signature is localised only in the library in plaintext form, which proves injudicious because it is not the library itself but the executable who presents infectious features. Secondly, the specific signature of this version, considered as out-dated, must have been suppressed for a virus however dating from 2004. The absence of signature in the database is confirmed by the negative results of every static analysis of the other versions.

Now that the generic signature has been localised, it becomes possible to use this information in order to consequently interpret the results of the dynamic analysis in an unbiased manner. A first interesting remark concerning the resident protection is that the antivirus does not block the installation of *MyDoom* and leaves the main process running even if it responsible for the propagation, the duplication, the residency and other infectious functionalities. It only warns of an infection as soon as the library containing the signature is extracted. This must be correlated with the negative result obtained when analysing dynamically the version without backdoor library ([FIN_NO_BDOOR]). This version is not detected though it implements every other infectious behaviour of the original strain. These results prove that the behaviors are simply undetected or remain ignored without the validation by means of a complementary signature. On the opposite, the other versions leaving this aspect of the final payload unmodified: the backdoor library ([DUP_SH_CUT, RES_WIN_INI]), are detected according to the same criteria than the original strain. These versions all contain the signature in their respective libraries proving once again our interpretation.

To go further, it is possible to localise more precisely the signature without performing a complete extraction like in [7]. It can be noticed that in the particular cases where the strings have been left as plaintext or encrypted by means of a stream cipher ([POL_PLAIN_STR, POL_FLOW_STR]), the library is not detected anymore. We can formulate the hypothesis that the signature is made of a shuffled string of the library or even of the shuffle mechanism itself. Once again, these two versions implement every infectious behaviour of the original strain, including the library extraction, but they remain undetected because they simply do not contain the signature. This conclusion backs up the fact that only the signature is taken into account in the context of dynamic analysis (Table 4).

Avast test results As far as Avast antivirus software is concerned, we first have a look on the static results and as a second step we will interpret the dynamic results paying particular attention to the versions which are not detected in static mode. The original *W32/MyDoom* strain is detected as "`Win32 : Agent - EZ[UnP]`". The first observation we can do is that the test where the malicious library has been removed from the virus ([CHARG_NO_BDOOR]) is negative. So the Avast! signature base contains a detection pattern located in the backdoor library of the worm. We can confirm this fact again, when we consider all the tests where the included library is left unchanged. These variants are all detected identically to the original *MyDoom* strain.

Table 4 Detection results with AVG

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	🟩	I-Worm/MyDoom
	shimgapi.dll library	I-Worm/MyDoom	N/A
FINAL	FIN_TRIG_TARG	🟩	I-Worm/MyDoom
	FIN_NO_BDOOR	🟧	🟧
DUPLI	DUP_SH_CUT	🟩	I-Worm/MyDoom
	DUP_NAM_PATH	🟩	I-Worm/MyDoom
RESID	RES_SERV_KEY	🟩	I-Worm/MyDoom
	RES_WIN_INI	🟩	I-Worm/MyDoom
POLYM	POL_FLOW_LIB	🟩	I-Worm/MyDoom
	POL_FLOW_STR	🟧	🟧
	POL_PLAIN_LIB	🟩	I-Worm/MyDoom
	POL_PLAIN_STR	🟧	🟧
ACTIV	ACT_EVENT	🟩	I-Worm/MyDoom
	ACT_MUTEX	🟩	I-Worm/MyDoom
OVINF	INF_DIF_KEY	🟩	I-Worm/MyDoom
	INF_SUP_HID	🟩	I-Worm/MyDoom
	INF_ENV_VAR	🟩	I-Worm/MyDoom

A distinctive feature that can be underlined for Avast is that the library tested alone and in plaintext form (deciphered) triggers a different detection alert "`Win32 : MyDoom - BJ [Wrm]`". So the signature database of Avast contains at least two different signatures for *MyDoom* corresponding to different versions of the worm. The first signature which states a generic version of the worm is, as we have seen above, located in the encrypted library. A single test where the strings have been left in plaintext ([POL_PLAIN_STR]) seems to contradict this. It should have been detected generically like the test where the strings have been encrypted by means of an alternative method (stream cipher) ([POL_FLOW_STR]). An explanation could be that by completely removing the ciphering system we have modified the location of certain bytes of the signature making it inefficient.

The second signature, which proved to be more accurate, is located in the library in plaintext (once deciphered) and raises the alert "`Win32: MyDoom-BJ [Wrm]`". In the test when the library has been left in deciphered form or encrypted by means of a stream cipher ([POL_FLOW_LIB]), the generic signature is no longer present; consequently Avast is able to detect the

Table 5 Detection results for Avast; * means identified as "Win32:Agent-EZ[Unp]" while ** means identified as "Win32:MyDoom-BJ[Wm]"

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	Agent-EZ*	Agent-EZ*
	shimgapi.dll library	MyDoom-BJ**	N/A
FINAL	FIN_TRIG_TARG	Agent-EZ*	Agent-EZ*
	FIN_NO_BDOOR	🟧	🟧
DUPLI	DUP_SH_CUT	Agent-EZ*	Agent-EZ*
	DUP_NAM_PATH	Agent-EZ*	Agent-EZ*
RESID	RES_SERV_KEY	Agent-EZ*	Agent-EZ*
	RES_WIN_INI	Agent-EZ*	Agent-EZ*
POLYM	POL_FLOW_LIB	🟧	MyDoom-BJ**
	POL_FLOW_STR	Agent-EZ*	Agent-EZ*
	POL_PLAIN_LIB	MyDoom-BJ**	MyDoom-BJ**
	POL_PLAIN_STR	🟧	🟧
ACTIV	ACT_EVENT	Agent-EZ*	Agent-EZ*
	ACT_MUTEX	Agent-EZ*	Agent-EZ*
OVINF	INF_DIF_KEY	Agent-EZ*	Agent-EZ*
	INF_SUP_HID	Agent-EZ*	Agent-EZ*
	INF_ENV_VAR	Agent-EZ*	Agent-EZ*

Table 6 Detection results with antiVirusKit 2006 – * means identified as "Win32.Worm.MyTob.2.Gen" (BD engine), ** means identified as "Email-Worm.Win32.MyDoom.Gen" (KAV engine) and *** means identified as "Trojan-Proxy.Win32.ProDoom.C (KAV engine)

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	MyTob.2.Gen*	MyTob.2.Gen*
	shimgapi.dll library	ProDoom.C***	N/A
FINAL	FIN_TRIG_TARG	MyDoom.Gen**	MyDoom.Gen**
	FIN_NO_BDOOR	MyDoom.Gen**	MyDoom.Gen**
DUPLI	DUP_SH_CUT	MyDoom.Gen**	MyDoom.Gen**
	DUP_NAM_PATH	MyDoom.Gen**	MyDoom.Gen**
RESID	RES_SERV_KEY	MyDoom.Gen**	MyDoom.Gen**
	RES_WIN_INI	MyDoom.Gen**	MyDoom.Gen**
POLYM	POL_FLOW_LIB	MyDoom.Gen**	MyDoom.Gen**
	POL_FLOW_STR	🟧	🟧
	POL_PLAIN_LIB	MyTob.2.Gen*	MyTob.2.Gen*
	POL_PLAIN_STR	MyDoom.Gen**	MyDoom.Gen**
ACTIV	ACT_EVENT	MyTob.2.Gen*	MyTob.2.Gen*
	ACT_MUTEX	MyTob.2.Gen*	MyTob.2.Gen*
OVINF	INF_DIF_KEY	MyTob.2.Gen*	MyTob.2.Gen*
	INF_SUP_HID	MyDoom.Gen**	MyDoom.Gen**
	INF_ENV_VAR	MyDoom.Gen**	MyDoom.Gen**

malware once the library containing the specific signature has been deciphered and extracted, only.

Now let us have a look on the versions which are not detected neither in static nor dynamic mode ([FIN_NO_BDOOR, POL_PLAIN_STR]). In these variants, as mentioned above, we have changed very few parts of the malicious code and especially we have not changed any of the main infectious behaviours from the original strain but they simply do not contain the signature anymore. This result is amazing because these modified malware work very well without generating any alert. Consequently, we can deduce that Avast is not able to detect them by observing the different behaviors of one of the most famous worms (Table 5).

G-DATA test results Each antivirus we have tested has its own features and original aspects. As far as GData's antivirusKit2006 is concerned they lie in the fact that this software uses two detection engines which GData has not created. It uses actually the KAV engine from the *Kaspersky Lab* company and the BD engine from *Bitdefender* antivirus from the *Softwin* company. The software enables the user to activate them either separately or together in parallel or sequentially. Our tests have considered this last setting. Let us make a first remark. According to the engine which has issued an alert, the worm is always detected in a generic way either as "`Email-Worm.Win32.My Doom.Gen`" or as "`Win32.Worm.MyTob.2.Gen`". This seems to indicate that behavioural detection is more seriously implemented in this software. Now when the strings are encrypted with a different encryption method than in the original strain ([POL_FLOW_STR]), the worm is not detected in any mode. We can deduce that either the original *ROT13* encryption mechanism or a jammed character string are the main clues that betray the infection.

The tests do not allow to understand why a particular engine is responsible for the detection of a given version and not the other one. Nevertheless it seems that the BD engine is more sensitive to the changes operated to certain keys of the registry base. As an example, instead of writing in a run key we have registered the virus in the file win.ini to achieve code residency ([RES_WIN_INI]), the detection does not occur anymore with respect to the BD engine but with respect to the KAV one. In the same way, for the overinfection test, if we keep on testing a registry key, even different ([INF_DIF_KEY]), the worm is still detected by the BD engine. Now, if we change the target of the test – a file, an environmental data ([INF_SUP_HID, INF_ENV_VAR]...) only the KAV engine still successfully detects it. To obtain more details with respect to this, we should have reconfigured differently the antivirus and activated separately each engine (Table 6).

From a general point of view, the results remains the same both in static and dynamic modes. Only a variant whose variable strings are encrypted by a different method ([POL_FLOW_STR]) – a stream cipher – is no longer detected. We can deduce that it is the only version affecting the integrity of the signature. This version is not detected neither in dynamic or static mode and yet it implements exactly the same behaviours than the original strain. We can thus consider that the behaviours either are not kept under supervision or they remain unconsidered without the validation by means of a signature.

Table 7 Detection results with F-Secure – * means identified as "Email-Worm.Win32.Mydoom.gen" and ** means identified as "Trojan-Proxy.Win32.Prodoom.c"

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	🟩	Prodoom.c**
	shimgapi.dll library	Prodoom.c**	N/A
FINAL	FIN_TRIG_TARG	Mydoom.gen*	Mydoom.gen*
	FIN_NO_BDOOR	Mydoom.gen*	Mydoom.gen*
DUPLI	DUP_SH_CUT	Mydoom.gen*	Mydoom.gen*
	DUP_NAM_PATH	Mydoom.gen*	Mydoom.gen*
RESID	RES_SERV_KEY	Mydoom.gen*	Mydoom.gen*
	RES_WIN_INI	Mydoom.gen*	Mydoom.gen*
POLYM	POL_FLOW_LIB	Mydoom.gen*	Mydoom.gen*
	POL_FLOW_STR	🟧	🟧
	POL_PLAIN_LIB	🟩	Prodoom.c**
	POL_PLAIN_STR	Mydoom.gen*	Mydoom.gen*
ACTIV	ACT_EVENT	🟩	Prodoom.c**
	ACT_MUTEX	🟩	Prodoom.c**
OVINF	INF_DIF_KEY	🟩	Prodoom.c**
	INF_SUP_HID	Mydoom.gen*	Prodoom.c**
	INF_ENV_VAR	Mydoom.gen*	Prodoom.c**

Table 8 Detection results with KAV – * means identified as "Email-Worm.Win32.Mydoom.gen" and ** means identified as "Trojan-Proxy.Win32.Prodoom.c"

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	🟩	Prodoom.c**
	shimgapi.dll library	Prodoom.c**	N/A
FINAL	FIN_TRIG_TARG	Mydoom.gen*	Mydoom.gen*
	FIN_NO_BDOOR	Mydoom.gen*	Mydoom.gen*
DUPLI	DUP_SH_CUT	Mydoom.gen*	Mydoom.gen*
	DUP_NAM_PATH	Mydoom.gen*	Mydoom.gen*
RESID	RES_SERV_KEY	Mydoom.gen*	Mydoom.gen*
	RES_WIN_INI	Mydoom.gen*	Mydoom.gen*
POLYM	POL_FLOW_LIB	Mydoom.gen*	Mydoom.gen*
	POL_FLOW_STR	🟧	🟧
	POL_PLAIN_LIB	🟩	Prodoom.c**
	POL_PLAIN_STR	Mydoom.gen*	Mydoom.gen*
ACTIV	ACT_EVENT	🟩	Prodoom.c**
	ACT_MUTEX	🟩	Prodoom.c**
OVINF	INF_DIF_KEY	🟩	Prodoom.c**
	INF_SUP_HID	Mydoom.gen*	Prodoom.c**
	INF_ENV_VAR	Mydoom.gen*	Prodoom.c**

F-Secure test results What is quite peculiar with F-secure is that the static sequence-based analysis method does not detect the original *W32/MyDoom* strain whereas others versions are detected as the generic worm "`Email-Worm.Win32.Mydoom.gen`". For example, when the backdoor library is removed ([FIN_NO_BDOOR]), although we have suppressed a whole part of the virus, it is detected as a generic version of *MyDoom*. This could be explained by the fact that a signature is a sequence of bytes which are supposed to be located at a given place in the executable code. If one byte is not present, the signature does not match (the detection function is in fact a simple AND function; see [7]). So whenever our modifications have changed either the value or the location of one of these bytes, the new version is detected or not. To understand this phenomenon we can compare the results between F-Secure and AntiVirusKit 2006 from GData. The similarity is striking and we can deduce that the signature database of F-Secure does not contain the "`Win32.Worm.MyTob.2.Gen`" signature relative to the Bit Defender engine, at all. This would explain why F-Secure antivirus does not raise any alert by manual scanning for the tests detected by the BD engine in AntiVirusKit 2006 ([INF_DIF_KEY, ACT_MUTEX, ACT_EVENT, POL_PLAIN_LIB]) and the original *Mydoom* worm (Table 7).

If we now focus on the dynamic detection tests, we obtain three different kinds of results. First, the versions which were detected with the generic signature "`Email-Worm.Win32.Mydoom.gen`" are also detected in the dynamic mode under the same label which proves that the signature has been involved in the process. Secondly we have the versions which are not detected in static mode but in the dynamic mode because of the specific signature which is in the extracted backdoor library ([ACT_EVENT, INF_SUP_HID]). You can notice that this signature is once again identical to the one detected with AntiVirusKit 2006 in the extracted library once deciphered. At last, remains the test where the library has been flow ciphered ([POL_FLOW_STR]) which seems not to contain any signature according to the F-Secure database and thus is not detected just like with AntivirusKit 2006. With regard to these similitudes, we can draw exactly the same conclusion as for AntiVirusKit, in F-Secure the signature remains mandatory independently from any behavioural detection.

Kaspersky Anti-Virus test results The analysis of the Kaspersky antivirus tests is going to be very short. The reader is invited to compare the result (Table 8) with the F-Secure results. This is not a copy/paste mistake but the exact result of the test. It has been proved recently in [7] that these two companies share the same signature databases. The exact similitude between the two is a proof that the behavioural engines which could have been different, have no influence at all on the results or that behaviour management, if any, is the same as well. Otherwise some slight variation should have been visible. The signature is the only element taken into account in the detection results.

DrWeb test results DrWeb antivirus introduces some other new particularities. The original *Mydoom* executable code is detected as "*BackDoor-Trojan*" during the static analysis and most of the modified variants show the same results except three of them. In the first undetected one, the worm duplicates itself under

Table 9 Detection results with DrWeb – * exact Dr Web antivirus message is "shimgapi.dll infected by Trojan-Proxy-470"

Behaviors	Versions	Static analysis	Dynamic protection
none	original strain	BackDoor-Trojan	Trojan-Proxy-470*
	shimgapi.dll library	Trojan-Proxy-470*	N/A
FINAL	FIN_TRIG_TARG	BackDoor-Trojan	Trojan-Proxy-470*
	FIN_NO_BDOOR	BackDoor-Trojan	🟧
DUPLI	DUP_SH_CUT	BackDoor-Trojan	Trojan-Proxy-470*
	DUP_NAM_PATH	🟧	Trojan-Proxy-470*
RES	RES_SERV_KEY	BackDoor-Trojan	Trojan-Proxy-470*
	RES_WIN_INI	BackDoor-Trojan	Trojan-Proxy-470*
POLYM	POL_FLOW_LIB	BackDoor-Trojan	🟧
	POL_FLOW_STR	BackDoor-Trojan	Trojan-Proxy-470*
	POL_PLAIN_LIB	MULDROP.Trojan	Trojan-Proxy-470*
	POL_PLAIN_STR	DLOADER.Trojan	BackDoor.Trojan
ACTIV	ACT_EVENT	BackDoor-Trojan	Trojan-Proxy-470*
	ACT_MUTEX	BackDoor-Trojan	Trojan-Proxy-470*
OVINF	INF_DIF_KEY	BackDoor-Trojan	Trojan-Proxy-470*
	INF_SUP_HID	BackDoor-Trojan	Trojan-Proxy-470*
	INF_ENV_VAR	BackDoor-Trojan	Trojan-Proxy-470*

a different path and more particularly in a folder compressed thanks to the NTFS mechanism ([DUP_NAM_PATH]). The compression must have made it undetected because we have tampered with the integrity of the signature. This signature seems also to be altered for the tests where the library and then the strings have been left in plaintext ([POL_PLAIN_LIB , POL_PLAIN_STR]), but for each of them, a new kind of alert is raised corresponding to different signatures. It is also interesting to notice that, still in the static mode, the signature "*BackDoor-Trojan*" does not seem to have any link with the backdoor library since the same alert is raised anyway for the version where the backdoor library has been removed ([FIN_NO_BDOOR]).

Let us move on to the dynamic mode. We have previously seen that the backdoor library alone is statically detected as "*Trojan-Proxy-470*". Most of the variants including the original strain are dynamically detected because of the extraction of *shimgapi.dll*. Let us notice that when the alert is triggered, the virus is already partially installed and DrWeb just offers to destroy the library *shimgapi.dll* which has just been extracted. On the other hand, two versions remain undetected in dynamic mode whereas they were in static mode ([FIN_NO_BDOOR, POL_FLOW_LIB]). DrWeb is the only antivirus presenting such results. It means that the resident protection does not use exactly the same detection engine than the manual scanning. If the others versions had been detected dynamically by behavioural analysis, these two versions should have been detected as well as they implement nearly the same behaviours. So resident protection mainly relies on signature as manual scanning did, except the fact that the detection pattern and/or the detection function differ (Table 9).

Table 10 Detection results with Viguard – * exact message is an alert of shimgapi.dll modification attempt; ** exact message is an alert to prevent the program from trying to set up on the start-up zone

Behaviors	Versions	1st execution	2nd execution
none	original strain	🟩	shimgapi.dll*
	shimgapi.dll library	N/A	N/A
FINAL	FIN_TRIG_TARG	In residence**	shimgapi.dll*
	FIN_NO_BDOOR	🟩	🟧
DUPLI	DUP_SH_CUT	🟩	shimgapi.dll*
	DUP_NAM_PATH	🟩	shimgapi.dll*
RES	RES_SERV_KEY	🟩	shimgapi.dll*
	RES_WIN_INI	🟩	shimgapi.dll*
POLYM	POL_FLOW_LIB	🟩	shimgapi.dll*
	POL_FLOW_STR	🟩	shimgapi.dll*
	POL_PLAIN_LIB	🟩	shimgapi.dll*
	POL_PLAIN_STR	🟩	shimgapi.dll*
ACTIV	ACT_EVENT	🟩	shimgapi.dll*
	ACT_MUTEX	🟩	shimgapi.dll*
OVINF	INF_DIF_KEY	🟩	shimgapi.dll*
	INF_SUP_HID	🟩	shimgapi.dll*
	INF_ENV_VAR	🟩	shimgapi.dll*

Viguard test results Up this point, the antiviral products we have tested show certain common features and approach but Viguard, in the antivirus classification, can be discarded. Its first well-known distinction is that it claims to not use any sequence-based detection technique except integrity checking. When installing Viguard antivirus, a fingerprint for the different files of the system is calculated and stored in a secure way. A drawback of this method – and it is clearly stated by Viguard – is that you need a safe and clean operating system without any malware before its installation.[6] A second major drawback is that it can generate a huge number of false positives. Indeed, the number of behaviours that are considered as potentially dangerous is particularly high. Most of the legitimate, uninfected programs adopt many of these behaviours.

Because of these particularities, we had to change the test procedures into a new protocol consisting in executing each variant twice without manual scanning.

During the first execution, we observe that no alert is raised for any malicious code except one ([FIN_TRIG_TARG]) for which Viguard warns us that a program is trying to set up in the start-up of the operating system (many other legitimate programs try to do that!). In fact, Viguard watches over specific zones of the registry base and triggers an alert whenever a program wants to register under a new or existing key.[7] With this approach,

[6] Indeed, we manage to install Viguard on a corrupted system and the malware has been certified without any alert. Consequently, the malware was able to run efficiently after the installation.

[7] Many of Viguard's alerts are rather useless for generic users who are unable to interpret them conveniently.

it is bound to generate a high number of false positives knowing that most of present legitimate programs do the same. It is even surprising to see that only this particular version is detected, the others trying to register under the same exact key. After this first alert, by choosing to keep the virus running, Viguard triggers several additional alerts warning the user that a "new e-mail program" is attempting to establish a network connection. Once again, this could be done by any e-mail client or other legitimate softwares.

For the second detection pass, in all the tests, Viguard warns the user that a program is trying to modify the existing file "*shimgapi.dll*". Viguard is not able to precise that it is a malicious file and would act the same way with any other file. In addition, it is left to the user to judge if the action is legitimate or not! We cannot really say that Viguard is implementing a real behavioral detection method. It simply watches disparate possibly malicious actions without correlating them (Table 10).

References

1. Beauchamp, K.G.: Applications of Walsh and Related Functions. In: Microelectronics and Signal Processing Series. Academic Press, ISBN 0-12-084180-0 (1984)
2. Chakrabarty, K., Hayes, J.P.: Balanced Boolean functions. IEE Proc. Comput. Digit. Tech. **145**(1) (1998)
3. Cohen, F.: Computer viruses. Ph.D. Thesis, University of Southern California, Janvier 1986
4. http://www.trendmicro.com/vinfo/virusencyclo/defaults.asp?VName=WORM_MYDOOM.A
5. Filiol, E.: Designs, intersecting families and weight of Boolean functions. In: Proceedings of the 7th IMA Conference on Cryptography and Coding. Lecture Notes in Computer Science vol. 1746. Springer Berlin Heidelberg New York (1999)
6. Filiol, E.: "Le ver MyDoom". J. Sécurité Informat. MISC **13** (2004)
7. Filiol, E.: Malware pattern scanning schemes secure against black-box analysis. J. Comput. Virol. **2**(1) (2006)
8. Filiol, E., Jacob, G., Le Liard, M.: Evaluation methodology of function-based malware detection. In: Proceedings of the First Workshop in Theoretical Computer Virology, Bonfante, G., Marion, J.- Y. (eds.) Nancy, May 2006
9. Jacob, G., Le Liard, M.: Evaluation des méthodes de détection comportementale des virus. Rapport de projet Mastère SSI, Laboratoire de virologie et de cryptologie et Supélec Bretagne (2006)
10. Josse, S.: How to measure the effectiveness of an antivirus. J. Comput. Virol. **2**(1) (2006)
11. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press, ISBN 0-8493-8523-7 (1997)
12. Morin, B.: Intrusion detection vs virology. In: Proceedings of the First Workshop in Theoretical Computer Virology, Bonfante, G., Marion, J.-Y. (eds.) Nancy, May 2006
13. Rothaus, O.S.: On bent functions. J. Combin. Theory **20**, 300–305 (1976)
14. Sperner, E.: Ein Satz über Untermengen einer endlichen Menge. Math. Z. **27**, 544–548 (1928)
15. Xiao, G.-Z., Massey, J.L.: A spectral characterization of correlation-immune combining functions. Trans. Inform. Theory **IT-34**(3), 569–571 (1988)