Using Infection Markers as a Vaccine against Malware Attacks

Andre Wichmann
Fraunhofer FKIE
Friedrich-Ebert-Alle 144
53113 Bonn, Germany
Email: andre.wichmann@fkie.fraunhofer.de

Elmar Gerhards-Padilla
Fraunhofer FKIE
Friedrich-Ebert-Alle 144
53113 Bonn, Germany
Email: elmar.gerhards-padilla@fkie.fraunhofer.de

***Abstract*—Malware is used by criminals for financial gains, espionage and sabotage, and their code and evasion techniques become increasingly complex and sophisticated. This means it takes longer for security researchers to analyse a malware and develop detection and removal routines, increasing the danger of critical systems becoming infected.**

In order to prevent multiple infections of the same system, malware often uses *infection markers* to mark a system as already infected. In this paper, we introduce the concept of using these markers to vaccinate systems against infections by a specific malware family. We discuss the characteristics of infection markers and develop a taxonomy of marker types. Then, we present a framework capable of classifying the infection marker used by a malware sample, and which can in most cases automatically extract the marker and generate a vaccination program. Evaluation with a large corpus of malware samples shows that for almost all malware that uses an infection marker, a vaccination program can be generated without the need of a human expert. Two case studies with prominent malware samples, *Sality* and *Conficker*, further show the potential of this approach.

***Index Terms*—malware; intrusion prevention**

I. Introduction

Malware, short for malicious software, is software that performs malicious actions on computer systems without the consent of their respective owners. While the first malware families have been created mostly as a proof of concept or to show off the author's technical skills, in recent years malware is created and spread by well-organized criminal gangs [1], [2], [3]. These cyber criminals use malware to steal credentials, send spam emails, conduct espionage, or sabotage computer systems. Since embedded devices have become ubiquitous in our modern society, these systems are vulnerable against these kinds of attack as well [4].

Malware is nowadays developed by professional and knowledgeable experts that go to great lengths to make it as difficult as possible for security software to detect and remove their product. Sophisticated stealth and obfuscation techniques are used to limit the effectiveness of modern anti-virus products. For example *Stuxnet*, a malware that was allegedly used to sabotage nuclear facilities [5], contains a database of common anti-virus products, and depending on the software that is installed on the target computer, takes measures to evade detection by that specific security product. Other malware is *polymorphic*, i.e. its code mutates between infections in order to evade signature-based detection. Even worse, on many embedded devices no anti-virus products even exist that could detect a malware attack.

Analyzing a new malware sample takes both time and expert knowledge. While some parts of the analysis process can be automated [6], [7], certain questions like how the malware spreads, what malicious functionality it contains or how it can be removed from an infected system are so open and general in scope that a human expert is required. In the case of *Stuxnet* it took several months before all infection vectors and its full malicious capabilities were discovered [5]. This means that for defense, the time between discovery of a malware and its analysis is critical. This is especially true for critical infrastructure, systems that are suspected to be targets of the malware, or environments where no anti-virus products exist. Protecting these systems from infections until reliable detection and removal procedures are developed is highly important.

In this paper, we propose a novel technique to quickly and reliably protect computer systems from infections by a known malware family. We exploit the fact that malware usually makes sure not to infect the same system twice by placing an *infection marker* on that system during the first infection. By injecting this infection marker into a clean system, it becomes *vaccinated* against this specific malware family. When the malware attacks the vaccinated computer, it will detect the infection marker and aborts the attack, believing it has already installed itself on that system.

This work makes the following contributions:

- The concept of *infection markers* is formally described and their characteristics are presented. A taxonomy of infection markers is developed, and the idea of using infection markers to prevent malware infections is introduced.
- A proof-of-concept framework for classifying and extracting infection markers from malware is presented. This framework is capable of automatically generating a vaccination software for protecting computers from infections by that malware, with no expert knowledge required.
- Using the infection marker taxonomy and applying the

above framework, an evaluation on a large corpus of malware samples is presented which provides an overview of the types of markers deployed by current malware.
- The effectiveness of using infection markers to protect systems against specific malware families is shown, both by the data generated with the framework and by two case studies with specific malware samples.

The rest of this paper is structured as follows. Section II explains what infection markers are, what types of markers exist, and how they can be used as a counter-measure against malware. A framework for classifying and extracting infection markers from malware is presented in section III, followed by an evaluation and two case studies in section IV. An overview of related work is given in section V, and a conclusion including a discussion on future work can be found in section VI.

II. Malware Infection Markers

From the point of view of a malware developer, it is very desirable that a computer system gets infected only once by that malware. Even if multiple variations of the same malware get released, as it is common among malware authors to evade signature-based detection [8], a system should not get infected a second time by the same malware family. Reasons for this are that multiple infections provide no advantage for the performance of the malware, but instead might even be dangerous to system stability of the attacked system because malware often nests deep inside the operating system.

It follows that it is worthwhile for malware developers to include a means to detect an already installed instance of their malware on a system to avoid duplicate infections. If a malware attacks a system and finds another instance of itself already there, it usually aborts the infection process and just exits. In order to achieve this, the first instance of the malware sets some kind of *infection marker* in the system which other instances can look for during infection. This infection marker can be a mutex in memory, the presence of a specific file on a storage device, a certain key in system registry or something completely different.

A. Infection Marker Characteristics

In order to work properly, an infection marker has to exhibit certain characteristics: It has to be *persistent* and *deterministic*.

Persistence in the context of infection markers means that it actually has to be present on the system at the time another instance of the same malware family tries to attack. This either means is has to be on a permanent medium, for example on the hard drive or the BIOS of a computer, or the malware has to set the marker anew each time the system starts if it is located in a temporary storage, for example in volatile memory.

Deterministic means that the location and the structure of the infection marker can be determined and read by all other instances of the malware during the infection process. A malware cannot simply generate a random number only known to itself as an infection marker, as a second instance of the malware would have no way of generating that same number to know what to look for. It can still be generated by a deterministic algorithm however, if it is based on parameters that will stay the same for all instances attacking a given system.

Apart from these two necessary characteristics, an ideal infection marker would also be **unique** in the space of malware family infection markers so that it will not collide with markers of other malware families. The presence of the marker from one malware family on a given system should not prevent another malware from successfully infecting that system. In addition, an infection marker should be **hidden** on the system so the user will not be alerted and does not notice the infection.

B. Using Infection Markers as a Defensive Measure

The concept of infection markers can be exploited to protect systems from getting infected by a specific malware. The infection marker can be set on a computer that is not yet infected, without installing the malware binary itself. When an instance of the malware attacks that computer, it will detect the infection marker and assumes the system is already infected and will not install itself. The system is *vaccinated* against that specific family of malware.

Using infection markers for vaccinating computer systems can be a very effective protective measure against malware that is otherwise good at evading detection by anti-virus solutions or for environments where no such software exists. Regardless of varying run-time packers [8] or polymorphism [9], by definition the infection marker has to remain constant for all versions of a malware family, thus becoming an invariant among all polymorphic versions of a malware family.

In order to be able to use an infection marker as a vaccine, it has to be determined how the marker looks like and where it has to be set on the system. Typically, reverse engineering has to be used to extract the infection marker. This is a complex and time-consuming process where an expert first has to remove all layers of protection from the malware before trying to locate this information inside the binary. Even though tools exist that can help an analyst [6], [7], reverse engineering tasks are often hard to automate completely because of the open nature of the problem. However, extracting infection markers can be different in that regard. We show in section IV that in practice the markers used by malware usually share some common characteristics. First, regardless of the marker type used, most of the time the marker is represented by a name, e.g. a mutex or file. Second, the creation of and the check for the infection marker usually goes through a confined, well-defined programming interface offered by the operating system. This makes it possible in many cases to fully automate the process of identifying and extracting the infection marker for a given malware sample, with no need for the labor or skill of a malware analyst.

C. Taxonomy of Infection Markers

Each type of infection marker has unique characteristics, which in turn has consequences for their use as a vaccine.

Because of this, we have developed a taxonomy of infection marker types that allows to classify malware based on different criteria.

1) Marker Location and Lifetime: There are different locations a malware can store its infection marker. Basically, any location that can be accessed by a program can be used: memory, the file system, or even the BIOS of the computer. The only limit is that an attacking instance of the malware has to be able to read from this location in order to detect the marker.

Marker locations can be characterized by the lifetime they offer: *Permanent* or *volatile*. An infection marker on a storage device or the BIOS can be considered permanent, while a marker in memory is volatile and will be gone when the system gets shut down. In the latter case, the malware will have to set it again when the computer starts the next time.

Common volatile infection markers on Windows are mutexes, named pipes or semaphores, but they could also be function hooks or bound network sockets.

2) Marker Type: As described in section II-A, an important property of an infection marker is that is has to be deterministic. Both *static* and *dynamic* markers fulfill this requirement.

A *static* infection marker means every instance of the malware uses the same marker for every computer it infects. On the other hand, a *dynamic* infection marker is computed on a per-system basis and will be different for each infected system. To satisfy the determinism property, an algorithm based on distinctive but permanent features of the infected computer is used to generate the infection marker. For example, *Conficker* uses the computer name as the base for an algorithm to derive a name for a mutex to use as its infection marker. To extract dynamic markers, the generating algorithm has to be reverse-engineered.

3) Coupling of the Marker with Malicious Functionality: An infection marker can be part of the functionality of the malware, or it can be something that is not related to how the malware works. Depending on how much the marker is coupled with the malware's functionality, using it as a vaccine can have side-effects which have to be dealt with.

Common infection markers like mutexes are not directly related to the malware's behavior. A vaccination program can easily create the mutex to protect a system against an infection. However, a malware could also use changes it makes to an infected system that have a direct impact on its malicious activities as an infection marker. For example, malware can hook into system API calls to modify the behavior of the operating system. If the malware used the presence of this hook as an infection marker, a vaccine would have to make sure that this hook does not lead to unwanted, malicious behavior.

4) Time of Marker Check: An infection marker's purpose is to prevent a second instance of a given malware from installing itself on a target machine. That means in principle it would be sufficient for the malware to check for the infection marker at the time of attack. In reality though, malware sometimes also checks for the marker each time it starts, for example because the marker is volatile like a mutex in memory and has to be set each time anew.

In this case, a vaccine can provide additional features. It then not only protects clean systems from becoming infected, it might also force the malware on an already infected machine to cease its malicious actions and become dormant. If at system startup the vaccination program manages to set the infection marker before the malware can, and if the malware checks for the marker each time it gets started, it will cease operation and become harmless. The *W32.Chydo* worm is an example for a case where this is possible. The system is still infected however, but this can be useful for example if there is no known reliable way yet to cleanly uninstall the malware.

Stuxnet would be a counter-example, as it uses a permanent registry key as its infection marker and only checks for it at the time of installation [5]. Only a vaccine for not yet infected machines would be possible.

5) Location of Marker Check: A malware infection of a computer system can involve several stages and several different programs, and thus the check for the infection marker can happen in any of these binaries. If the user gets tricked into deliberately running the malware via social engineering or in the case of trojans, the main malware binary will be installed on the attacked computer immediately. However, in cases like self-propagating worms or attacks via drive-by downloads, shellcode or a dropper might get executed first that then downloads the main malware binary over the network to install it. Once the malware has successfully infected the machine, it might download other, additional modules from its command-and-control server.

The fact that the check for the infection marker can happen in any of the involved binaries has direct consequences on the task of extracting the marker. It can be easier to get access to a specific piece of malware than to get hold of its dropper that might have deleted itself already. If the infection marker is checked in the dropper binary during the time of installation, the main malware binary will not contain information about the marker and thus it cannot be extracted from there for use as a vaccine.

III. Marker Classification and Extraction Framework

In order to find out if a given malware uses an infection marker and to classify its type, we have created a framework that processes malware samples and in most cases is capable of automatically generating a vaccination program that can be used to protect computers against that malware. This is a proof of concept for Microsoft Windows malware, but the underlying concept can be adapted to other architectures as well.

A. Architecture

The analysis framework consists of several components. The *Process Observer* is a program that monitors the malware process and its activities, capable of following the malware code into other processes it might inject itself into. It logs all user-level API calls related to mutexes, files, registry

keys, named pipes, and mailslots, as they could be possible candidates for setting and reading infection markers. Each time such a function gets called, the name and parameters of the accessed object is logged.

The *Controller* processes a database of malware samples and uses four virtual machines running Windows XP with the Process Observer component installed. Two runs are needed to decide if an infection marker is used, and another two runs to detect the type and test if the malware is susceptible against vaccination. This vaccination test is performed using the *Marker Injector*, which is capable of creating mutexes, files, named pipes, mailslots, and registry keys. Figure 1 depicts the workflow of a malware sample analysis run.

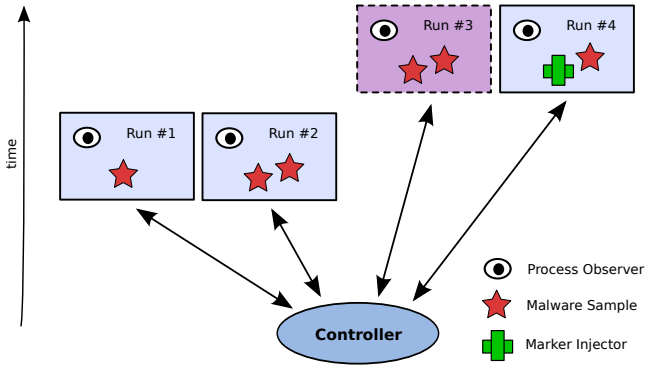

Fig. 1. Workflow of testing the presence and type of infection markers

For each malware sample, four analysis runs are conducted.

- **Run 1:** The malware gets observed for two minutes. The output of this run is used as a reference for the other runs.
- **Run 2:** The malware gets started normally, without observing its behavior. After 30 seconds, a second instance of the malware gets started under control of the Process Observer. If the malware uses an infection marker to prevent multiple installations, the events observed during this run should be different from those of run 1. In particular, the number of logged events and the running time of the malware program should be much lower, as the malware will exit after detecting that the first instance has already infected the machine. If the use of an infection marker is detected, run 3 and 4 are initiated.
- **Run 3:** The setup for run 3 is similar to that of run 2, only that several parameters of the system environment are configured differently. The goal of this is to determine if the infection marker is static or dynamic. To this end, many parameters that might be used in the generation of dynamic infection markers are changed: the computer name, the user's name, the MAC address of the network interface, the ID of the hard drive, the serial number of the installed operating system, the language, and the locale. If the infection marker differs from run 2, it is assumed the marker is dynamic, otherwise it is likely a static marker.
- **Run 4:** From comparing the results of run 1 and run 2, candidates for possible infection markers are generated by looking at the last API calls the malware has made before exiting in run 2. For run 4, the Marker Injector component is used to set the candidate infection markers. Then, the malware is run under surveillance of the Process Observer. If the candidates are the real infection markers, the number of observed events and the running time of the sample should be similar to that of the second sample in run 2. If the candidates did not contain the correct infection markers, the log should be similar to that of run 1 instead.

B. Limitations of the Framework

Even though the framework is capable of checking for the use of infection markers in malware and can even extract the used marker in most cases, it has some limitations, some of them inherent in the technology used.

First, at the heart of the framework lies the technique of dynamic analysis, i.e. the malware gets executed and monitored in a secured and controlled environment. Malware can detect this environment [10] and behave differently, or thwart the analysis by going into a state of inactivity for some time during startup. However, this is a problem all dynamic analysis techniques have to face.

Second, the heuristics used in the framework are not guaranteed to produce accurate results. The heuristic to distinguish between static and dynamic markers might fail if the malware uses a characteristic which is not different between the machines of run 2 and 3. In addition, only markers that use the APIs mentioned above can be identified. If malware deploys more low-level techniques that bypass these API calls, the framework will fail to detect the type of infection marker even though it will still be able to detect that a marker is used.

IV. Evaluation

To evaluate the concept of using infection markers as a counter-measure against malware, two different questions have to be examined. The first is how many malware samples use infection markers and what types, and the second is how many of the samples that use infection markers are susceptible to vaccination. To answer these questions, a corpus of current malware samples was evaluated using the framework presented in section III, and two high-profile malware samples were examined as case studies.

A. Malware Corpus

A corpus of 1496 randomly selected malware samples that have been collected over the course of two years between September 2009 and September 2011 by honeypots, spam traps and user submissions was used in the evaluation. As it turned out, 889 (59.4%) of all samples use some kind of infection marker (c.f. table I). No statement can be made about the other 40.6% that did not exhibit this behavior. It is not clear if they really do not use an infection marker or if they detected the analysis environment, were not compatible with

Total number of samples	1496
Samples using an infection marker	889 (59.4%)
Correctly identified markers	847 (95.2%)

TABLE I
USE OF INFECTION MARKERS

it, or checked for the marker only after the analysis time of two minutes.

At first glance, 59.4% seems a low value for the percentage of malware that prevents multiple infections. However, this can be put into perspective by the fact that malware often comprises more than one binary and that the marker check can happen in any of them (c.f. section II-C5). The malware corpus contains droppers and malicious helper modules downloaded by the main malware samples, and an infection marker is only used in *one* of the multiple binaries related to a specific malware. Analysis of a few random samples from the set of malware that does not seem to use infection markers supports this hypothesis. Unfortunately, it is not practical to reverse-engineer all samples, so we cannot say for sure and this question remains open for future research.

Still, the majority of samples indeed use some kind of infection marker *and* are susceptible to a vaccination counter-measure. From all these 889 samples, for 847 samples (95.2%) the analysis framework was able to detect and extract the infection marker correctly. Only for 42 out of 889 samples it was not possible to draw any conclusions about the type of infection marker used. Either a type not covered by the analysis framework was used, or the check for the marker happened not as one of the last actions before exiting, for example because the malware modified some files or registry keys after the check.

Table II shows the number of static and dynamic markers used for the samples where the marker could be correctly identified.

Samples using a known infection marker	847
Static markers	842 (99.4%)
Dynamic markers	5 (0.6%)

TABLE II
STATIC VS. DYNAMIC MARKERS

The overwhelming majority of samples, over 99%, seem to use a static infection marker. The dynamic markers detected were a named pipe and two mutexes that included varying numerical values. The distribution of the marker types among all the successfully identified markers is listed in table III.

Samples using a known infection marker	847
Mutexes	834 (98.4%)
Registry keys	9 (1.0%)
Named pipes	3 (0.3%)
Files	1 (0.1%)
Mailslots	0 (0.0%)

TABLE III
MARKER TYPES

What can be seen is that with 98.4%, the vast majority of infection markers are mutexes. Only a very small number of registry keys (9), named pipes (3) or files (1) and no mailslots are used. That means that in almost all cases, the infection marker can be used as a vaccine without any side-effects (c.f. section II).

B. Case Studies

While a quantitative evaluation of a large corpus of malware is insightful with regards to what portion of the general malware population is susceptible to vaccination and what types of infection markers are used, it is also interesting to examine some well-known, sophisticated malware families which are considered to be above-average dangerous. To this end, we have evaluated two additional malware samples which rank 2^{nd} and 3^{rd} on the list of malware most seen in the wild in the *Symantec Intelligence Report February 2012* [11] (the first rank is occupied by a generic detection heuristic): *Sality*, and *Conficker*.

Conficker is a highly sophisticated worm using several different attack vectors for infecting computer systems. Even though the command channel has been disrupted for years now, millions of systems are still infected. Our framework correctly determines that it uses a mutex as an infection marker, and that it is susceptible to using a vaccine to protect clean computers. It also correctly identifies that the mutex is dynamic, which means it cannot automatically generate a vaccination program. However, the information revealed by the framework gives an excellent starting point to speed up the reverse engineering process for extracting the algorithm for the dynamic mutex.

Sality is a file-infecting virus which incorporates a complicated polymorphic engine that mutates its code and makes each infection look differently [12]. Our framework correctly determines that the infection marker is a static mutex. This allows it to automatically generate a vaccination program that can be used to protect not yet infected systems.

V. RELATED WORK

Almost no scientific literature on malware mentions infection markers. [13] discusses different mitigation techniques for the *Conficker* worm, focusing on providing network detection signatures and ways to remove the worm. They also present the idea of using the infection marker as a counter-measure. However, no generalization of the concept is provided, no investigation of different types of markers is done, and all the findings are made by using manual reverse engineering with no automation.

Most methods to prevent infections on the host rely upon detecting the malware first in order to prevent or undo an infection. The first techniques for malware detection used byte signatures [14], [15]. However, mutating the data of malware binaries via packing [8] or polymorphism [9] makes these methods less effective. In contrast, our approach utilizes an invariant that has to be stored persistently on an infected system and which cannot be mutated by the malware.

Later approaches use static analysis of code which incorporates the semantics of the program [16], [17], [9]. Unfortunately, these techniques assume that the relevant binary

code can be accessed and disassembled correctly. Runtime packing, self-modifying code and using techniques like opaque constants can effectively thwart static analysis [18].

Because of the limits of static analysis, other research focuses on employing dynamic analysis to detect and prevent malware infections on the host. In [19], a malware's behavior is monitored in a sandbox during a preparation phase, and a behavior graph is generated which models relations between system calls together with their parameters. Then, on the host to be protected, running processes are monitored and their behavior is matched against the generated signature graphs to decide if a process is malicious or not. Similar approaches can be found in [20] and [21].

Besides possible false negatives and the extra runtime overhead, these approaches rely on the problematic assumption that the process under observation does neither detect nor escape the analysis. In addition, until it is declared malicious, the process might already have caused harm which cannot easily be undone, like sending data over the network. In contrast, the concept of using infection markers as a vaccine is a purely passive approach. While the Process Observer in our proof of concept framework can be evaded by malware as well, more sophisticated sandbox technologies like [6] could be employed to improve resilience, something that cannot be done with behavior analysis frameworks on the end host without adding even more overhead.

VI. Conclusion

In this paper, we have introduced the concept of using infection markers, which get used by malware to prevent multiple infections of the same computer, to protect clean systems from getting infected. The advantage is that infection markers are not affected by traditional evasion techniques like code obfuscation or polymorphism, and that in most cases the extraction process can be fully automated. This makes it a promising approach for a first line of defense after discovering a new piece of malware and for protecting critical systems against specific malicious threats, especially for environments where no security software exists yet.

A proof-of-concept framework has been presented that can determine the type of marker used by a given malware, and in most cases is even capable of automatically extracting it. Evaluation with a corpus of current malware and with two case studies of well-known malware families shows that for malware binaries that make use of infection markers, the overwhelming majority use static mutexes which can easily be extracted and used as a vaccine without any side-effects.

To further the understanding of how and where malware uses infection markers, more research on the relationship between different binaries of a malware is needed. In addition, dynamic markers, even though they seem to be rare at the moment, can only be detected, but not yet automatically extracted. Combining a less invasive analysis environment like [6] with approaches that are capable of extracting whole algorithms from binaries [22] could make vaccination more applicable to an even wide range of malware.

References

[1] T. Holz, M. Engelberth, and F. Freiling, "Learning more about the underground economy: A case-study of keyloggers and dropzones," *Computer Security–ESORICS 2009*, pp. 1–18, 2009.

[2] B. Stone-Gross, T. Holz, G. Stringhini, and G. Vigna, "The underground economy of spam: A botmasters perspective of coordinating large-scale spam campaigns," in *USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2011.

[3] B. Chu, T. Holt, and G. Ahn, "Examining the creation, distribution, and function of malware on-line," Technical Report for National Institute of Justice. NIJ Grant, Tech. Rep., 2010.

[4] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 447–462.

[5] N. Falliere, L. O. Murchu, and E. Chien, "W32.Stuxnet dossier," Symantic Security Response, Tech. Rep., Oct. 2010.

[6] A. Dinaburg, P. Royal, M. Sharif, and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 51–62.

[7] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," in *Proceedings of the 2nd USENIX conference on Large-scale exploits and emergent threats: botnets, spyware, worms, and more*. USENIX Association, 2009, pp. 8–8.

[8] F. Guo, P. Ferrie, and T. Chiueh, "A study of the packer problem and its solutions," in *Recent Advances in Intrusion Detection*. Springer, 2008, pp. 98–115.

[9] F. Leder, B. Steinbock, and P. Martini, "Classification and detection of metamorphic malware using value set analysis," in *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 2009, pp. 39–46.

[10] X. Chen, J. Andersen, Z. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*. Ieee, 2008, pp. 177–186.

[11] Symantec, "Symantec intelligence report: February 2012," Symantec, Tech. Rep., 2012.

[12] N. Falliere, "Sality: Story of a peer-to-peer viral network," Symantic Security Response, Tech. Rep., Jul. 2011.

[13] F. Leder and T. Werner, "Know Your Enemy: Containing Conficker, To Tame a Malware," The Honeynet Project, http://honeynet.org, Tech. Rep., Apr. 2009.

[14] P. Szor, *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[15] A. Sung, J. Xu, P. Chavez, and S. Mukkamala, "Static analyzer of vicious executables (save)," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 326–334.

[16] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, "Semantics-aware malware detection," in *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005, pp. 32–46.

[17] C. Kruegel, W. Robertson, and G. Vigna, "Detecting kernel-level rootkits through binary analysis," in *Computer Security Applications Conference, 2004. 20th Annual*. IEEE, 2004, pp. 91–100.

[18] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 421–430.

[19] C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in *Proceedings of the 18th conference on USENIX security symposium*. USENIX Association, 2009, pp. 351–366.

[20] B. Rozenberg, E. Gudes, Y. Elovici, and Y. Fledel, "A method for detecting unknown malicious executables," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 190–196.

[21] T. Nykodym, V. Skormin, A. Dolgikh, and J. Antonakos, "Automatic functionality detection in behavior-based ids," in *MILITARY COMMUNICATIONS CONFERENCE, 2011-MILCOM 2011*. IEEE, 2011, pp. 1302–1307.

[22] C. Kolbitsch, T. Holz, C. Kruegel, and E. Kirda, "Inspector gadget: Automated extraction of proprietary gadgets from malware binaries," in *IEEE Symposium on Security and Privacy*, 2010, pp. 29–44.