Ontology for Malware Behavior: a Core Model Proposal

André Grégio*†, Rodrigo Bonacin*‡, Olga Nabuco*, Vitor Monte Afonso†, Paulo Lício de Geus† and Mario Jino†
*Center for Information Technology Renato Archer (CTI) — Campinas, São Paulo, Brazil
Email: {andre.gregio, rodrigo.bonacin, olga.nabuco}@cti.gov.br
†University of Campinas (Unicamp) — Campinas, São Paulo, Brazil
Email: {vitor, paulo}@lasca.ic.unicamp.br, jino@dca.fee.unicamp.br
‡Faculty of Campo Limpo Paulista (FACCAMP)
R. Guatemala, 167, 13231-230, Campo Limpo Paulista, São Paulo, Brazil

***Abstract*—The ubiquity of Internet-connected devices motivates attackers to create malicious programs (malware) to exploit users and their systems. Malware detection requires a deep understanding of their possible behaviors, one that is detailed enough to tell apart suspicious programs from benign, legitimate ones. A step to effectively address the malware problem leans toward the development of an ontology. Current efforts are based on an obsolete hierarchy of malware classes that defines a malware family by one single prevalent behavior (e.g., viruses infect other files, worms spread and exploit remote systems autonomously, Trojan horses disguise themselves as benign programs, and so on). In order to address the detection of modern, complex malware families whose infections involve sets of multiple exploit methods, we need an ontology broader enough to deal with these suspicious activities performed on the victim's system. In this paper, we propose a core model for a novel malware ontology that is based on their exhibited behavior, filling a gap in the field.**

***Keywords*-Malware; Ontology; Computer Security**

INTRODUCTION

Internet-connected devices are now ubiquitous, turning their users into targets of all sorts of attacks. The main threat faced by information systems and their users today is malicious software (malware). Current malware samples are very complex pieces of software that leverage a broad range of techniques to attack computer systems. These attacks aim to compromise systems in a way that the malware can install itself, keep its access, bypass the security mechanisms, and finally, accomplish its objective. Cyber criminals write malware with several purposes, such as of stealing information, making money, attacking other systems etc. Since a malware sample is a program to be executed in a victim's system, the cyber criminal purposes are translated into instructions performed during the infection process. These instructions, or actions, define the "behavior" of a malicious program and may serve to measure the extent of the caused damage.

Malware behavior analysis is an important step to help in the process of discovering suspicious patterns, which can then be used in detection procedures. Behavior-based malware detection is a delicate task, since it depends on the monitoring of a victim's system during the infection. Monitoring requires to capture interactions between the malware process (or processes) and, more importantly, to filter those that are relevant for security. This involves an "a priori" knowledge about what can be considered as suspicious behavior, as well as how the interactivity occurs among the objects of a malware infection. Since a target system runs several processes that may be legitimate or not, we need to gain knowledge about these processes' behavior to decide whether they are suspicious or not, thus requiring further, deeper analysis. Hence, there is a need to define the scope of suspicious behaviors, and how collected information about the interactions among processes and a target system can help in the identification of malware infections.

One step toward a solution involves the proposition of an ontology to handle suspicious behavior and, consequently, malware. Malicious programs are traditionally classified into well-known categories according to a predominant behavior (infects a file, spreads autonomously, disguise itself as a legitimate program), such as virus, worm, and Trojan. Moreover, existing malware ontologies lead to an hierarchy of classes (or categories, such as virus, worm, Trojan etc.) that are far from adequate to address the issue of modern malware, which are multipurpose, complex, and may be split among several components. In addition, Obrst et al. [1] mention that those class-based malware ontologies may "not be useful for malware instances that exhibit either behaviors from multiple classes or novel behaviors not associated with any recognized class".

Therefore, there is an urgent need for an ontology that addresses behavioral aspects and activities performed by malicious programs. In this paper, we propose a novel malware ontology based on a set of commonly observed suspicious behaviors. Our main contributions are two-fold:

1) We define a hierarchy of main behaviors, each one consisting of a set of suspicious activities. The goal is to identify unknown malware according to its exhibited behavior.
2) We propose an ontology that models our knowledge on malware behavior, which can serve as a basis for future developments on reasoning and detection procedures.

I. Background

Ontologies are flexible structures, both human and machine interpretable, which can be populated by instances creating a knowledge base, or can describe a process, or even can be a common language for a community or enterprise, and also several others possibilities. In our case, it can be a software structure that identifies and interpret an unusual behavior performed by some known process. It is possible, due to ontologies inherent inference capabilities, to categorize if a behavior is suspicious even if we dont have a pattern describing it. Ontologies have languages with specific expressivity to describe hierarchies, relationship between classes, multiple inheritance, and describe characteristics through the use of axioms. Another richness of ontology languages can also be given by joining rule languages and/or query languages to the main ontology, for instance, providing the link between the open world provided by ontologies and the closed world provided by rule languages, thus making the ontology less undecidable.

A. Developing the ontology: methodology

There are several methodologies to develop an ontology. Generally speaking, most of them recommends using scenarios, making competency questions, reusing others' work, applying divide and conquer methods, etc. For the best, divide the ontology into subdomains would help to discover related works and patterns, easing the work to be done. Scenarios describe possible applications of the ontology that lead to the construction of competency questions, which are questions (very related to the subject and recursive) that the user wants the ontology to answer. Hence, there is no real sequence in building an ontology, since the process involves a back and forth approach, which is not only possible, but even desirable to understand the domain and its representation. Also, prototyping and making short evaluations use to work for most of them.

In this development, OWL 2 (Ontology Web Language) was used to express the actions performed by a malicious software. In our model we used four attributes to express behaviors: the source object, which is an application process; the action performed by this application process (e.g., write, delete, create, and others); the object that is the target of the software (e.g., a registry, a file, a process); and the identification of the application that is suspect to be the victim of the attack. Figure 1 illustrates our model.

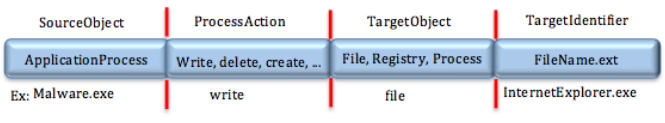

Figure 1. Attributes used to express behaviors

B. Related work

One of the main motivations for ontologies regarding malware detection is the need of mechanisms that are able to address unknown threats and whose operation do not rely solely on signatures (which need to be manually crafted as new malware samples arise). However, most of the related works lack details about their proposed ontologies and are based on obsolete malware classes (virus, worm, Trojan) that are not well suited to address the current, complex and multi-behavior malware samples seen in the wild.

TWMAN (Taiwan Malware Analysis Net) is a platform composed of three layers—knowledge, communication, application—that are designed to provide a knowledge and rule base, an ontology language, and reports about analyzed malware samples [2]. TWMAN dynamic analysis component runs malware samples and monitors their activities during the infection. The monitored activities are those related to changes in the value of some Windows Registry keys (Run and Service), network traffic and its associated information (IP addresses, application layer protocols), and files added, modified or deleted in the victim's system (accomplished with the use of the AIDE tool [3]). The authors present the structure of their malware-related ontology without discussing it any further. The structure is composed of the object "Thing", which has four elements: "Malware_Impact_Target", "Malware_Type", "Malware_Behavioral", and "Malware_Sample". The description of each element is not provided, whereas "Malware_Impact_Target" can be "Network", "Registry" or "File", and "Malware_Type" is limited to "Trojan", "Worm" or "Backdoor". Their example is very simple and the proposed structure lacks precision to deal with the complexity of malware samples found in the wild. In addition, TWMAN assumption on malware types may not provide useful results considering the myriad of malware classes (not addressed) and their behavioral combinations.

Martnez et al. propose uCLAVS, a an architectural model for malware detection that is based on a malware intrusion ontology [4]. Their purpose is to detect malicious contents in a file by submitting it to multiple antivirus engines for analysis in a Web service fashion. The detection results produced by the antivirus are used as prevention rules in their proposed ontology, whereas other attacks were created using testing tools. Thus, this ontology represents signatures for some modelled network attacks and for malware detected by antivirus. The authors defined inference rules to handle the ontology's attributes, such as "Malware-Behavior", which only shows whether a file is detected as malware or not by some of the antivirus engines present in uCLAVS. However, any more details about their ontology are not provided and their detection approach is not new, since publicly available services (e.g., VirusTotal - http://www.virustotal.com) allow any user to scan a file using more than 40 antivirus.

Tafazzoli and Sadjadi propose a malware ontology based on fuzzy logic [5]. The superclass of their ontology is called "Malact", classified as "artifact" or "non-artifact". The group of "artifacts" includes the following malware types: virus, worm, botnet, spyware, backdoor, Trojan horse, rootkit and exploit. The authors describe four types of characteristics and their values, regarding malware objectives, behavioral and technical features from an operational perspective, malware architecture and placement on the target (centralized, distributed, local, remote), and malware communication/management. Each type of malware is associated with the values of the four described characteristics, producing a scheme that define those types. The authors mention the relation among their chosen malware types as an example of use of their ontology. Although their proposed ontology goes one step further in behavioral aspects of malware in comparison to the other related works, it defines a smaller, more general set of behaviors (e.g, unauthorized access and use, disclosure, etc.), which is based on malware classes that comprise a single predominant behavior.

Obrst et al. propose the development of an ontology for the cyber security domain to integrate data from different sources [1]. Their purpose is to reuse existing ontologies related to the information security domain (attacks, vulnerabilities, malware) and is based on the diamond model of malicious activity. This model considers each of the four corners as a dimension of a threat (victim, infrastructure, capability and actor). To address malware, they rely on the work of Swimmer [6], which describes a malware class hierarchy based on known and traditional malware classes. As we stated before, Obrst et al. also noticed that this type of classification is problematic to address complex malware: modern samples cannot be tied to standard classes since they exhibit multiple and distinct behaviors. Other related approaches include MAEC (http://maec.mitre.org) and the MAL [7], a language to describe malware in details and a malware-related dictionary of terms, respectively.

II. The conception of the ontology

The ontology conception followed the reasoning of representing the behavior of malicious software. We had to identify the traces of malware in order to build classes and their relationships. It was made upon scenarios, not only competency questions, describing a possible attack over the Microsoft Windows system. The creation of these scenarios make it possible to identify the main attributes of a representative knowledge base of actions, behaviors, target systems, source systems and the possible sequence of occurrence of this behavior. Ontologies can turn this task into a loosely coupled process, keeping the suspected instance in halt and relating the behavior to several suspected process at same time, even if it is only partial. It means that it can be captured even if it is not completely similar to the process previously described. Once identified a possible infected process due to its suspicious behavior, one can keep it as an instance of a distinct process and resume the analyses, instead of discard it because it doesnt really fit any previously described behavior.

A. Suspicious behavior

The behavior of any program can be considered as the set of actions it performs during execution; in case of malware, the entire behavior is suspicious, since each of the performed actions may be a step of an infection procedure [8]. Thus, we propose that a given program behaves suspiciously if it presents one or more of the six "events" described below. Each one of them consists of several activities that can be observed during the analyzed program's execution.

1) Attack Launching: Malware samples may use the infected computers to launch attacks against other systems. The attack launching event includes:

Denial of Service (DoS). Violates the availability of a victim's system by draining its resources (e.g., network bandwidth, memory, processing power).

E-mail Sending. E-mail content can be a massive amount of unsolicited messages (spam) or specially crafted ones to spread malware by luring users into following links or executing attachments (phishing).

Scanning. The victim's recognition by the mapping of systems and services through the network.

Exploit Sending. The main method for worms spreading. If a system is found vulnerable (through scanning), an attacker or malware sample sends a piece of code specially crafted to get access into this system, exploiting it.

2) Evasion: Occurs when a malware sample wants to prevent from being inspected by an analysis tool (or forensics expert) or tries to bypass the system's security mechanisms.

Anti-Analysis. Dynamic malware analysis tools are usually deployed in virtual or emulated environments in order to be scalable. This happens due to the large amount of new malware variants that are discovered daily and the need to analyze as many of them as possible, since they are available. Thus, malware authors often embed anti-analysis techniques within their samples before distributing them, so as to avoid that such analysis tools monitor the actual malware behavior. Common anti-analysis techniques are described as follows.

- **Debugger Checking:** a debugger tool may be used to discover, step by step, the instructions performed by a malicious program. Malware can then inspect whether some flag is set or not, such as "IsDebuggerPresent", to finish their own execution.
- **Environment Detection:** virtual and emulated environments have innate fingerprints (e.g., the inability to perform certain instructions, a specific name for a device, the way to handle the network interface settings) that can be used by a malware to detect if it is running inside an analysis environment. If the sample detects an analysis environment, it usually quits.

- **Removal of Evidence:** after executing (or applying some anti-analysis) technique, malware may remove its own binary file from the infected system to avoid being caught by a forensic analyst. Before deletion, however, malware may infect other files or disguise itself to remain active in the victim's system.

Anti-Defence. Modern operating systems deploy basic security mechanisms by default, such as safe-mode execution, firewall and automatic updates. In addition, users of these systems may have additional security mechanisms, e.g., an antivirus. Some malware samples apply anti-defence techniques to bypass those security mechanisms and accomplish the intended infection. Anti-defence techniques include:

- **Removal of Registries:** specific registry entries are set when a system needs to reboot in safe-mode. This mode is commonly used in administrative repairs, as well as to check for security issues in a possibly compromised system. There are malware samples that remove these registry keys in order to avoid that the system enters in safe-mode, which could hinder the malware sample correct functioning.
- **Shutdown of Defence Mechanisms:** protection measures (system firewall, automatic updates, antivirus) can be turned off by certain malware as a first execution action. This can be done by either editing registry keys related to these services or terminating the service running process. Thus, malware samples are able to freely communicate through the network, to be active since no further update would crash them, and to remain undetected by the installed antivirus.

3) Remote Control: There are malware types whose acts rely on sending and receiving orders through the network. These communication allows them to be remote controlled by an attacker, to receive other pieces of malicious code, and even to be updated.

Download Code. Some attacks rely on multistage infections: first, a program is inserted and executed in the victim's system to serve as a "downloader", i.e., it obtains the actual malware. Usually, downloaders prepare the environment and fetch the piece of code that will pursue the attackers objective. It may be a standalone, known malware, or another code that may, for instance, substitute a system's library.

- **Known Malware Execution:** (almost) harmless malware may download and execute an already known malware with specific features (e.g., backdoor, virus, rootkit, worm) required to complete the infection.
- **Other Code Execution:** besides known malware, other types of codes may be downloaded and executed in a compromised system: 0-day malware, i.e., a sample that has no detection procedure, system libraries, data files to be used by the running malware, Trojanized applications, and so on.

Get Command. The difference between the previous behavior, download code, to this one is that the download action is active, whereas get command involves that a sample waits for receiving an external order. Types of passive interactions include the following.

- **Binary File:** a remote command may be issued so that the infected system get a binary file (that can be a picture, for example) that contains embedded orders. Other examples of binary files include Web pages (HTML files), torrents and executables.
- **Configuration File:** malware can receive orders to get a new configuration file, which can point them to another command and control center or to change a potential denial of service target. Configuration files serve to the purpose of updating the malware sample once it is already in place.
- **IRC/IM Connection:** the most common method to get commands from an attacker is to connect to an Instant Relay Chat (IRC) or Instant Messaging (IM) server. In general, commands (often encrypted) are issued in the conversation topic or chat's subject.

4) Self-Defence: A self-defence event occurs when malware samples try to protect themselves against analysis and detection, as well as from other external interactions, such as other malware samples, incompatible operating system, reboots and power shortages, etc. Two types of behavior are described as Self-Defence events, anti-analysis, which is also an evasion technique, and maintenance.

Anti-Analysis. This behavior is the same previously described as part of the "Evasion" event and contains the same set of suspicious activities (see Section II-A2).

Maintenance. This event consists of procedures to assure that a sample will effectively infects the target system and survive potential adversities.

- **Component Checking:** Some malware samples require additional components to run or to compromise the machine. For instance, a sample may require a version of the FlashPlayer application, or the Java Virtual Machine, or that an application is installed and configured (Outlook Express with the contacts list). Checking for the presence of a required component is then necessary to prevent malware crashes.
- **Create Synchronization Object:** a synchronization object, often called a mutex, is usually created to "announce" the presence of a program, or to "lock" some resource. Malware samples use to create specific mutexes to prevent that the same malware instance (from another attack) reinfects the machine and crashes the victim's system.
- **Language Checking:** lazy malware developers write their programs with hard-coded paths, demanding verification on the compromised system's language. In other situation, malware whose targets are well defined

(directed attacks) should check the victim's language to avoid running in systems outside of the infection's scope.

- **Persistence:** this activity intends to assure that a malware sample will remain active after a reboot or a shutdown without requiring exploiting the target again. The persistence is accomplished through changes in the system settings that force the malware to run when the system initiates.

5) Stealing: Depending on the malware's motivation, it will need to steal system and/or user data to accomplish its goal (fraud, identity theft, impersonation of a system or a user). We divided this behavior into two events according to the type of stolen data, system or user information.

System Information Stealing. This type of stealing may be used to impersonate a machine or to gather knowledge about a system that can lead to further attacks. Pieces of system information that can be stolen include hostname, operating system and resources in general.

- **Hostname:** the name associated with the machine is also its network name and may reveal, for example, the local group in which it belongs.
- **OS information:** attacks usually relies on the type and current version of the operating system. This information allows an attacker or a malware sample to know if a certain patch/update is installed.
- **Resources information:** processing power, memory capacity, and hard disk space are important information to run specific types of malware. For instance, a sample whose target is a smartphone needs this kind of information to decide if it is able to run, whereas more complex malware (for firmware) may need specific information about the serial number of the targeted device to complete the infection.

User Information Stealing.

Attackers capture user information for diverse activities, commonly the credentials, which allow for the access in remote devices and services, as well as Internet Banking account data, which may lead the user to financial losses.

- **Credential:** monitors such as keyloggers capture credential data, i.e., usernames and passwords that allow for further access to e-mail/e-commerce accounts, remote login on other systems, and so on.
- **Internet Banking Data:** the capture of Internet Banking data is broader than credentials, it also involves stealing the values present in additional authentication factors, such as password tables and hardware tokens. With this data in hands, an attacker can transfer money from the victim's account, do online shopping, etc.

6) Subversion: It is common for malicious software to subvert the operating system and its applications in order to change the victim's standard behavior and to remain unaware. Those changes affect the system operation in an overall manner, as we can notice in the described activities.

Browser. Subverting the browser is usually accomplished through the install of a plug-in, add-on or extension. Thus, the browser can be reconfigured to inadvertently point the user to a malicious location. Another browser subversion technique is to load a proxy auto configuration file (PAC) to resolve target domains into compromised IP addresses.

Memory Writing. This is a technique widely used to subvert system programs and user applications. A malware sample injects itself (or other malicious code) into a target application by writing in the target's memory space. Thus, malware can control those compromised processes and propagate without the need of its original process. This technique is also known as process hijacking.

Operating System. Subverting the operating system is an effective way to dominate the system. One of the most dangerous subversion technique is to load a kernel driver that modifies the operating system behavior in a way that it can prevent it (or additional protection mechanisms) to find malware files, processes and even network communication. A simpler way to subvert the operating system is to change its "hosts" file. Hence, when a user's application looks up to resolving network names (e.g., Internet Banking domains), it is inadvertently redirected to a malicious site.

III. Ontology-based cyber security model

The proposed ontology is based on the conceptual aspects presented in the last section. Figure 2 presents an overview of the Malware Ontology, i.e., the core classes, which we expanded to specialized ones, the relationships and data properties. In the following paragraphs, we describe the main concepts and relationships (the class names are in italics).

The *SuspiciousExecution* is a central class of the MalwareOntology. It is associated with a *SuspiciousSoftware* that is executed on a *System*. Each *SuspiciousExecution* has a set of *ProcessAction*s. Each *ProcessAction* instance is related to a *SourceObject*, such as an *AplicationProcess* that executes it. A *ProcessAction* is also related to a *TargetObject* that can be *Mutex*, *Network*, *Registry*, *SystemFile* or another *AplicationProcess*. A process' *ProcessAction* is identified with a unique name/identifier. The process actions are also associated with timestamps. Finally, when applicable, a process can be linked to a *SuspiciousBehaviour* instance, which can be one or more of the following events: *AttackLaunchingEvent*, *EvasionEvent*, *RemoteControlEvent*, *SelfDefenceEvent*, *StealingEvent*, or *SubversionEvent*.

Figure 3 illustrates the expanded subclass *EvasionEvent* of the *SuspiciousBehaviour* classes hierarchy, according to the conceptual framework described in the last section (II-A2). This event consists of two main behaviors, *AntiAnalysis* and *AntiDefence*, which are then specialized in suspicious activities. These activities can provide more details about the actions performed, as is the case of *ShutdownDefenceMechanisms*. Axioms and properties complement this model.

Figure 2. Overview of the Malware Ontology

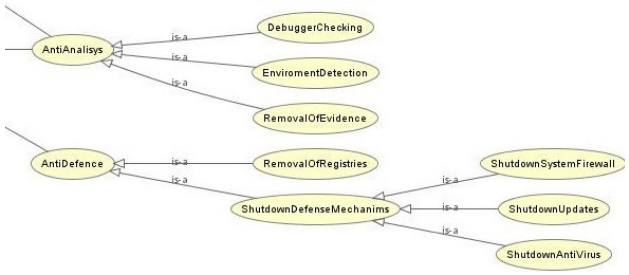

Figure 3. Expanded view of "EvasionEvent".

To illustrate our ontology, we analyzed a malware sample (`mw.exe`) that, among other "legitimate" activities, performed the three actions below:

1) mw.exe; write; file; c:\windows\...\javaservice.exe
2) mw.exe; setvaluekey; registry; ...\run\javaservice.exe
3) mw.exe; delete; file; c:\mw.exe

Mapping these logs to our ontology leverages that: `mw.exe` is an *ApplicationProcess* from the *SourceObject* class; `write`, `setvaluekey`, `delete` are instances of *ProcessAction*; `file` and `registry` are instances of *TargetObject* whose values are the action targets `c:\windows\...\javaservice.exe`, `...\run\javaservice.exe`, and `c:\mw.exe`. Notice that the target of item 3 is also an *ApplicationProcess*. Moreover, item 1 exhibited a suspicious behavior of *RemoteControlEvent* (*DownloadCode⇒OtherCodeExecution*), item 2 exhibited a *SelfDefenceEvent* through a *Maintenance* behavior named *Persistence*, and item 3 exhibited an *EvasionEvent* through an *AntiAnalysis* behavior named *RemovalOfEvidence*.

A set of process actions with suspicious behaviors is the basis for modelling inference rules to determine if an instance of *SuspiciousExecution* is linked to a malware sample. This work is limited as it does not provide a ready to use solution, once an extensive empirical work must be done to define the rules and other parameters to automatically detect potential malware with high level of precision. However, we expected to contribute with an ontological framework that is able to categorize and represent suspicious behavior in a precise way, which is a fundamental step for further developments on reasoning and detection procedures.

IV. Conclusion

In this paper, we presented an ontology based on a set of suspicious behaviors observed during malware infections on victim's systems. The differences among our proposed ontology and existing ones is that it is not tied to traditional malware classes, but to potentially dangerous behaviors. Therefore, we are able to identify unknown programs as malware. As a future work, we will define inference rules to apply our ontology on malware detection procedures.

References

[1] L. Obrst, P. Chase, and R. Markeloff, "Developing an ontology of the cyber security domain." in *STIDS*, ser. CEUR Workshop Proceedings, P. C. G. da Costa and K. B. Laskey, Eds., vol. 966. CEUR-WS.org, 2012, pp. 49–56.

[2] H.-D. Huang, T.-Y. Chuang, Y.-L. Tsai, and C.-S. Lee, "Ontology-based intelligent system for malware behavioral analysis." in *FUZZ-IEEE*. IEEE, 2010, pp. 1–6.

[3] R. Lehti, P. Virolainen, R. van den Berg, and H. von Haugwitz, "Advanced intrusion detection environment," http://aide.sourceforge.net.

[4] C. A. Martinez, G. I. Echeverri, and A. G. C. Sanz, "Malware detection based on cloud computing integrating intrusion ontology representation," in *Communications (LATINCOM), 2010 IEEE Latin-American Conference on*, September 2010.

[5] T. Tafazzoli and S. H. Sadjadi, "Malware fuzzy ontology for semantic web," *International Journal of Computer Science and Network Security*, vol. 8, pp. 153–161, 2008.

[6] M. Swimmer, "Towards an ontology of malware classes," http://www.scribd.com/doc/24058261/Towards-an-Ontology-of-Malware-Classes, 2008.

[7] D. A. Mundie and D. M. Mcintire, "The mal: A malware analysis lexicon," http://www.sei.cmu.edu/reports/13tn010.pdf.

[8] A. R. A. Grégio, V. M. Afonso, D. S. F. Filho, P. L. Geus, M. Jino, and R. D. C. Santos, "Pinpointing malicious activities through network and system-level malware execution behavior," in *Computational Science and Its Applications ICCSA 2012*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7336, pp. 274–285.