Pinpointing Malicious Activities through Network and System-level Malware Execution Behavior

André Ricardo Abed Grégio[1,2], Vitor Monte Afonso[2], Dario Simões Fernandes Filho[2], Paulo Lício de Geus[2], Mario Jino[2], and Rafael D. C. dos Santos[3]

[1] Renato Archer IT Research Center (CTI/MCT), Campinas, SP, Brazil
argregio@cti.gov.br
[2] University of Campinas (Unicamp), Campinas, SP, Brazil
{vitor, dario, paulo}@las.ic.unicamp.br, jino@dca.fee.unicamp.br
[3] Brazilian Institute for Space Research (INPE/MCT), S. J. dos Campos, SP, Brazil
rafael.santos@lac.inpe.br

Abstract. Malicious programs pose a major threat to Internet-connected systems, increasing the importance of studying their behavior in order to fight against them. In this paper, we propose definitions to the different types of behavior that a program can present during its execution. Based on those definitions, we define *suspicious behavior* as the group of actions that change the state of a target system. We also propose a set of network and system-level dangerous activities that can be used to denote the malignity in suspicious behaviors, which were extracted from a large set of malware samples. In addition, we evaluate the malware samples according to their suspicious behavior. Moreover, we developed filters to translate from lower-level execution traces to the observed dangerous activities and evaluated them in the context of actual malware.

Keywords: computer security, malware analysis, behavioral traces

1 Introduction

Malicious software are a major threat to Internet-connected systems. This kind of software ranges from worms and trojan horses to rootkits and botnets and is generically referred to as "malware". Thousands of malware variants arise periodically, hindering their analysis and the creation of effective vaccines by antiviruses companies. Publicly available dynamic analysis systems (e.g., Anubis [12], CWSandbox [17], Norman [1], ThreatExpert [2]) provide reports that give an overview of a malware sample behavior. However, they present too many technical details and/or too much information in a slew of activities that may confuse a user on finding the activities that characterize the malignity of an analyzed sample.

We propose a simpler and focused approach to describe malicious activities that is based on the higher-level behavior extracted from analyzed malware samples. Thus, we can bridge lower-level and specialized actions, such as a kernel function call or a write operation performed into a specific registry key, to understandable, identifiable high-level activities that emphasizes only the suspicious behavior. This can be useful to allow the identification of malware variants, to speed up incident response and to help in the development of malware removal procedures.

The main contributions of this article can be summarized as follows:

– We introduce a new notion of "execution behavior", splitting it in subsets according to the kind of interference perceived on the target system—active, passive and neutral–modelling a program's behavior in a simplified way.
– We characterize suspicious behavior by narrowing the scope of malware behavioral analysis to a reduced set of actions that change the state of a system.
– We define a "knowledge base" of network and system-level actions that correspond to intelligible activities (behavioral filters) that extends the set of behaviors described on the field's literature.
– We developed a prototype tool that is added in our dynamic analysis system to apply the behavioral filters and automatically extract potentially dangerous activities (i.e., suspicious behaviors) from the execution trace of a malware sample.

Additionally, we tested our proposed behavioral filters on a large set of actual malware samples that were gathered from malware collection honeypots [15] and spam attachments and then executed in our dynamic analysis system, BehEMOT [3]. At the end of this process, we pinpoint the malicious activities obtained from these samples and leverage results that allow us to analyze nuances among malware from different sources and with distinct assigned labels.

2 Related Work

In [9], the authors propose a theoretical model to perform behavior-based detection of infectious actions. Their work presents a strong mathematical basis to define different types of behavior and is an extension of a previous work that was only able to handle sequences of bytes [8]. The limiting factor is that their new approach requires the malware's source code, which is sometimes difficult to obtain.

The authors of [11] describe a malicious behavior model that is based on attribute grammars. They propose an abstraction layer to bridge the semantic gap between the behavior-based detection of malware and subtleties of platforms and systems. They trace a behavior by executing the sample inside a virtual machine and monitoring its system calls, which are then translated to their malicious behavior language. They examine and formally define four types of malware behavior.

It is worth to note that, in our approach, we do not consider the malware sample's source code, as it would require a decompilation step that could turn into an impossible task, due to the use of packers. Therefore, our work is based solely on the behavior logged during the execution of a program, which can change due to peculiarities of the monitoring environment [4].

In [14], the authors propose to bridge the semantic gap using behavioral graphs that were manually built to correlate common actions found in malicious bots, such as e-mail sending and data leaking. They evaluate seven kinds of behaviors related to bots, thus their coverage is mostly based on network actions.

Bayer et al. [13] provide a view on different behaviors presented by almost one million malware samples that were analyzed by Anubis over 2007 and 2008. They produced statistics and analyze trends, showing the percentage of observed samples that performed a variety of actions, from a simple file creation to the installation of a Windows kernel driver. We took some of their observed behavior to compose our suspicious activity definitions, but instead of analyzing the overall scene we tried to delve into behaviors that we believe that pose more risk to target systems.

Although the literature's research works are very rich in their definitions and findings, we believe that there is still a lack of focus on the practical applicability of dynamic analysis systems as true generators of heuristics to detection schemes. In addition, previous works on the subject take only specific malware classes into account, e.g., spyware [7], bots [14], worms/viruses [11]. In this paper, we propose to provide high-level behavioral filters to find suspicious behaviors bound to malware samples independently of their assigned classes.

3 Behavioral Traces

To the extent of this work, we use behavioral traces to pinpoint the security-relevant activities that a program performs. In this section, we explain how we extract a malware's execution trace to identify suspicious activities. Also, we define different types of "behavior" within the scope of this article.

3.1 Extraction and Processing

The first step to extract a behavioral (execution) trace from a malware sample is to run it inside a controlled environment and to monitor the important security-related actions performed during a limited execution time. As the prevalent kind of current malware targets Microsoft Windows-based systems, we chose them as the main focus of interest. Hence, we developed a framework [3] to capture some selected system calls through SSDT hooking [10] and to translate them to high-level actions, logging the produced trace.

The monitored system calls considered in this paper are related to operating system's entities—file, process and registry—that cover a great variety of potentially suspicious actions and are registered in a trace. This trace is, in its raw form, an ordered set of the performed system calls and their parameters, which need to be processed to represent a higher-level behavior. To monitor a malware sample's network behavior, we developed a layer driver[4] which is interposed be-

[4] Layered drivers are kernel modules that can manipulate data flows from the operating system through a specific driver, such as a device driver.

tween the network interface driver and the Microsoft Windows operating system. Thus, our layer driver is able to capture all network operations that a malware sample performs during its analysis and that are directed to the drivers that control the devices related to TCP and UDP communication. Therefore, it is possible to obtain network connections and attempts to open local ports made during the analysis time.

To process a behavioral trace we need to translate the monitored system calls into meaningful actions. This is done to facilitate the interpretation of the extracted behavior, as in some cases more than one system call may represent a single operation. Each action is represented by a number of attributes: the **timestamp**, to identify the action's position in the chain of captured events, the **source** process, which represents the performer of an action, the **operation**—i.e., the type of interaction, like CreateProcess, DeleteFile and ConnectNetwork—between the source and the **target** of that operation.

To illustrate this process, let's suppose a program "*mw.exe*" that wants to create a process "*mwproc.exe*". To do this, it calls the `ZwCreateProcess` routine. When this happens, our tool intercepts the system call and produces an action formatted in the following way:

```
<ts>,C:\mw.exe,CreateProcess,C:\mwproc.exe
```

In the same way, the routines `ZwSetValueKey` and `ZwDeleteKey` are translated to the `WriteRegistry` and `RemoveRegistry` operations, respectively.

The advantage of processing system calls in the above way is that when several routines serve to the same purpose, we map them to a single operation. For example, if an action's goal is to delete a file, this can be accomplished by `ZwOpenFile`, `ZwDeleteFile` or `ZwSetInformationFile` with carefully crafted values as their parameters. By abstracting from the particular variant chosen by a monitored program, we are able to present a much more meaningful result, i.e., a `DeleteFile` operation.

3.2 Definitions of Behavior

The general behavior of a program consists of the set of actions performed during its execution by an operating system. In the previous section we defined "action" based on some attributes (timestamp, source, operation, target). Thus, an action "α" is a tuple composed by the values of the aforementioned attributes and can be represented as $\alpha = \{ts, src, op, tgt\}$. Therefore, to define a behavior we proceed as follows:

Let B be the general behavior of a malware sample M_k, and A^{M_k} be the set of N actions α_i performed during its execution, so that $A^{M_k} = \{\alpha_i\}_{i=1..N}$ and $B(M_k) = A^{M_k}$.

The set of actions that compose a behavior can be divided into groups according to their nature: if an action interferes with the environment, i.e. changes the state of the system, it is part of an **active** subset of the behavior. This is the case of actions that involve a file write, delete or creation, for instance. Otherwise, the action is **passive**, meaning that it gathered a piece of information without modifying anything, for example, read, open or query something.

However, there is a subset of the general behavior that is **neutral**, i.e., the actions can be either active or passive, but they do not lead to a malign outcome. The neutral behavior contains common actions that are performed during a normal execution of any program, such as to load standard system libraries, to read or to configure registry keys and to create temporary files.

3.3 Suspicious Behavior

When a malicious program is executed, each of its actions can be considered suspicious. These actions constitute a suspicious behavior that, when analyzed, may reveal important details related to the attack. For instance, a malware sample that downloads another piece of malicious code and use it to spread itself has to perform a network connection, to write the file containing the malicious code on the compromised system and to launch the process of the downloaded file that will handle the spreading process.

Therefore, we are only interested in actions that modify the state of the compromised system (the active subset of the behavior, that is, B_A) at the same time that we want to avoid the actions that are considered normal to a program's execution (the neutral behavior, that is, B_N). Thus, we define the suspicious behavior of a malware sample M_k as $B_S(M_k) = B_A(M_k) - B_N(M_k)$. From the analysis of each obtained $B_S(M_k)$, we extract a set of network and system-level actions that represent dangerous activities to the security of a system.

4 Malicious Activities

During execution, a software piece interacts actively and passively with the operating system. Thus, benign software presents active behavior such as creating new registries, writing values to registry keys, creating other processes, accessing the network to send debug information or to search for updates, downloading and writing new files etc.

Therefore, as any piece of software does, malware interact with the operating system in the same way. However, malware interactions cause undesired changes on the operating system settings. These changes must be detected to allow for a damage report and to begin an incident response procedure.

Hence, it is necessary to pinpoint the actions that correspond to dangerous or malicious activities, so as to allow better understanding of the malware diversity. To do this, we defined an initial set of network and system-level activities that present a certain level of risk and that can be obtained from selected actions extracted from the suspicious behavior.

4.1 Network-level Risky Activities

Evasion of Information. Information related to the operating system or the user can be evaded through the network, such as the hostname, hardware data, network interface data, OS version and credentials. An adversary may use this information to choose targets for an attack, or to map her compromised machines (e.g., zombie computers that are part of a botnet). In a directed attack, sensitive documents may be stolen and transferred to an FTP server, for instance. Also, information can be evaded through a POST (HTTP method) performed on a compromised server, a FTP transfer, an SQL update query to a remote database or an e-mail message sent through an open SMTP server.

Scanning. Worm-like malware need to perform scans over the network to find possible targets for spreading. This involves the search of known vulnerable services or unprotected/open network applications. Apart from spreading, a malware sample may also perform scans to find out a network topology or to find trampoline systems that could be used to launch attacks anonymously.

DoS. There are classes of malware (e.g., bots) whose features include flooding attacks to perform denial of service (DoS). This is oftenly done through the sending of an overwhelming amount of UDP packets, for example, by the nodes (infected machines) of a botnet.

Downloading. Some types of malware are composed by several pieces that execute specialized tasks. Thus, the first piece—the downloader—is responsible for downloading the other components, such as libraries, configuration files, drivers or infected executable files. This compartimentalization is also used by malware developers to try to avoid antiviruses or other security mechanisms. This activity can also indicate a drive-by download, which is a download commonly performed during a user's Web browsing without his/her knowledge.

E-mail Sending. A malicious program can communicate with its owner through e-mail to announce the success of an attack or to send out sensitive data from the compromised machine. Also, a compromised machine can be used as an unsolicited e-mail server, sending thousands of spam on behalf of an attacker that is being paid for the service. The victim's machine is "rented" and acts as a provider of spam or phishing, aiming to distribute commercial messages or even malicious links or attached infected files [6].

IRC Connection. If an attacked system becomes part of a botnet, it needs to "phone home", i.e., to contact a C&C[5] server to receive commands, updates etc. Botclients commonly connect to an Instant Relay Chat (IRC) server that acts as a C&C.

[5] Command and Control that manages the bots that belong to a botnet.

4.2 System-level Risky Activities

Name Resolution File Modification. A trojan-like malware sample can modify the network name resolution file to forward users to a compromised server and lure them into supplying their data. These can be credentials (e-mail, online banking, Web applications such as Facebook and Twitter) or financial information (credit card numbers). This kind of modification tricks users to access fake online banking sites as a direct cause of malware known as “bankers”.

Evidence Removal. Some malware disguise themselves as system processes to deceive security mechanisms or forensic analysis: they can “drop” a file that was embedded in a packed way inside their main file or download the actual malicious program from the Internet. In some cases, these droppers/downloaders remove the evidence of compromise, deleting the installation files after the attack. In addition, a malware sample that is able to identify that it is being analyzed may also remove itself from the system.

Critical Registry Key Removal. There are registry keys that are critical to the normal operation of a system, e.g., the one that allows initialization in secure mode. The removal of this kind of key can cause instability in the system and inconvenient obstacles during a disinfection procedure.

Security Mechanisms Corruption. To compromise a target system while avoiding detection, malware authors usually try to identify and disable security mechanisms. This activity can be accomplished by turning off the system firewall or known antivirus engines, through the termination of their processes and/or removal of the associated registry keys.

Browser’s Proxy Modification. The effect of this activity is similar to that described on “name resolution file modification”. The difference here is that a malware sample loads a cofiguration file in the browser’s memory (when it is running) that changes the proxy on the fly, resulting in an automatic redirection of the user to a malicious site. Malware usually do this using PAC (proxy auto-config) files, which are effective on different operating systems and browsers.

Driver Loading. Drivers are kernel modules that access the most privileged level of a system. A driver makes the interface between the operating system and the hardware, such as network interfaces, graphic cards and other devices. However, drivers are also used by rootkits, a kernel-level kind of malware that can hide their processes, files and network connections in order to remain undetected.

5 Experimental Results

We collected 1,641 malware samples from July, 2010 to July, 2011—463 from honeypots (**collector**), 1,182 from spam messages (**phishing**)—and executed them in our dynamic analysis environment, which is a Qemu-emulated [5] MS Windows XP SP3. This execution produced a behavior trace for each analyzed sample, which were also sent to the VirusTotal service (http://www.virustotal.com) so that we could get their Kaspersky Anti-Virus (KAV) label. We then applied the behavioral filters described in Section 4 to the traces and analyzed the network- and system-level malicious activities. We discuss, the observed malicious activities over the full malware set in Section 5.1 and the results regarding different malware classes (attributed by KAV) in Section 5.2.

5.1 Malicious Activities' Pinpoint

The purpose of the behavioral filters is to map suspicious actions performed by a program to intelligible activities. These filters provide high-level and useful information about a malware sample execution and describe its presented behavior. For example, if a malware sample tries to turn off the security mechanisms natively running on a Windows OS to avoid detection and weaken the machine defenses, it commonly launches a script that performs some shell commands, such as `net stop ''Security Center''`, `net stop SharedAccess` and `netsh firewall set opmode mode=disable`. Also, a sample might perform changes on the `FirewallPolicy\StandardProfile` registry keys, for instance, by setting the value of **EnableFirewall** to “0”.

This kind of action causes a positive match against our behavioral filter and leverages, in the particular aforementioned example, “Security Mechanism Corruption” as a malicious activity found in the evaluated sample. When the “pinpointing” process is finished, we have a list of dangerous (and potentially malicious) activities for each sample from our malware dataset. This process produced the results from Table 1, divided by source (phishing or collector), when applied to the complete dataset.

Table 1. Malicious activities discovered through the pinpointing process on our collection; sum may be higher than 100%.

Level	Activity	Phishing (%)	Collector (%)
Network	NT1 (Evasion)	3.72	6.69
	NT2 (Scan)	14.21	50.54
	NT3 (DoS)	37.22	29.37
	NT4 (Download)	1.10	9.07
	NT5 (E-mail)	1.95	3.45
	NT6 (IRC)	0.42	9.28
System	OS1 (Hosts File)	1.10	0.43
	OS2 (Evidence)	15.06	4.32
	OS3 (Critical Key)	0.34	0.43
	OS4 (Security Bypass)	4.48	5.18
	OS5 (PAC)	0.25	0
	OS6 (Driver)	5.16	0

We notice that most of the analyzed malware samples performed the same set of malicious activities, despite their source. Those activities are attempts to scan networks for vulnerable services or UDP flooding (NT2, NT3) at the network-level and self-removal and security mechanism bypass (OS2, OS4) at the system-level. From the samples that came from our collectors, 15.15% did not present any behavior, either due to crashing during the execution, to corrupted binaries or to the use of anti-analysis techniques. From the samples obtained by e-mail crawling (phishing set), 12.01% either presented an incomplete trace or did not match any of our defined suspicious behaviors.

5.2 Malware Classes Behavior

As mentioned previously, we obtained the KAV labels from VirusTotal for each malware sample from our dataset. Then, we processed these labels to extract only the assigned class (e.g., trojan, worm, backdoor etc) according to the Kaspersky naming rules [16]. These rules define a naming system that is composed by `[Prefix:]Behavior.Platform.Name[.Variant]`, where the parameter *Behavior* represents the malware class.

Hence, we grouped those samples whose assigned class is the same and analyzed the ten more populated classes, which correspond to more than 85% of the samples (excluding the $\approx 7\%$ that are unknown to antivirus engines at the time of this analysis, i.e., August, 2011). After that, we tested our behavioral filters on each sample of the ten selected classes to extract their malicious activities (defined in Section 4).

We show the network-level malicious activities performed by the different classes in Table 2 and the system-level ones in Table 3, where a checkmark (✓) denotes that at least one of the samples assigned to the class (rows) performed the suspicious activity (column) and a blank entry denotes that this behavior was unmatched.

Table 2. Union of network-level risky activities per malware class.

Class	**NT1**	**NT2**	**NT3**	**NT4**	**NT5**	**NT6**
Worm	✓	✓	✓	✓	✓	
Backdoor	✓	✓	✓	✓	✓	✓
Trojan	✓	✓	✓	✓		
Downloader	✓		✓	✓	✓	
Virus		✓	✓			
UNKNOWN		✓	✓			
Packed			✓			✓
Gamethief			✓			
Banker	✓		✓		✓	
Dropper	✓		✓		✓	

Table 3. Union of system-level risky activities per malware class.

Class	**OS1**	**OS2**	**OS3**	**OS4**	**OS5**	**OS6**
Worm	✓	✓	✓	✓		✓
Backdoor	✓	✓		✓		✓
Trojan	✓	✓	✓	✓		✓
Downloader	✓	✓		✓		✓
Virus		✓		✓		
UNKNOWN		✓		✓		✓
Packed		✓		✓		✓
Gamethief		✓		✓		✓
Banker	✓			✓	✓	
Dropper	✓	✓		✓		✓

These tables are the union of the pinpointed malicious activities from a specific AV-assigned class, i.e., if at least one sample from the assigned class performed the activity, then we put a checkmark. The ideal situation happens when a malware class is characterized by a specific behavior, for instance, a downloader obtains something from the Internet and a worm tries to spread. Although this may be true, antivirus labels are not good to separate malware in meaningful, representative classes as the assigned name can confuse and mislead the user about the actual behavior of a sample.

Thus, if we take a closer look on the tables' results, it is worth noting that there are classes whose samples share a great amount of suspicious activities among each other. To illustrate this, lets analyze the three most populated classes: worm, backdoor and trojan. From Table 2, it seems that worms differ from backdoors only by "NT6", whereas trojans differ from worms by "NT5" and from backdoors by "NT5" and "NT6". However, due to the fact that our results are presented as the union of identified suspicious activities, there could be samples from these three distinct classes that share a common subset of network and system-level presented behavior. One such instance might happen when some samples from the worm, backdoor and trojan classes perform exclusively the suspicious network-level activities "NT1", "NT2", "NT3" and system-level activities "OS1" and "OS2". Therefore, although these samples are classified by KAV into three distinct classes, if we consider their observed behavior they should be assigned to a single one. Unfortunately, all antivirus engines share the same problem, making their malware assigned labels nearly useless to users when regarding the malicious behavioral information.

Conversely, our approach produces detailed enough information that can provide a better understanding about the risks related to a program's execution. It is also possible to overcome the misclassification of antivirus engines by classifying the unidentified (UNKNOWN) samples by their traced behavior. This way, our scheme can be used to generate a malware class characterized by"NT2", "NT3", "OS2", "OS4" and "OS6", thus avoiding the false-negative results produced by antivirus engines.

It is interesting to notice that in Table 3, as expected from malware that attack online banking sites, only the bankers presented the behavior labeled as “OS5” (Browser’s Proxy Modification).

6 Conclusion

In this paper, we divided a program’s execution trace in different types of behaviors and proposed the suspicious behavior definition to denote the dangerous activities that change the state of a system. We leveraged behavioral filters composed by these activities—performed at the network and system-level—to identify potentially harmful actions and to help with incident response as well as to provide a better understanding of malware. To evaluate our approach, we tested it in a dataset of malware collected from different sources. We provided results that show the percentage of malware samples that presented our behaviors and that compare them to AV-assigned classes. This latter comparison pointed to the problems in the current malware naming scheme, which we plan to address in a future work.

References

1. Norman Sandbox. `http://www.norman.com/security_center/security_tools/`
2. ThreatExpert. `http://www.threatexpert.com/`
3. Afonso, V.M., Filho, D.S.F., Grégio, A.R.A., de Geus, P.L., Jino, M.: A hybrid framework to analyze web and os malware. In: Proceedings of the 2012 IEEE International Conference on Communications (ICC) (June 2012)
4. Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G.: Efficient detection of split personalities in malware. In: 17th Annual Network and Distributed System Security Symposium (NDSS 2010) (2 2010)
5. Bellard, F.: Qemu, a fast and portable dynamic translator. In: USENIX Annual Technical Conference, FREENIX Track. pp. 41–46 (2005)
6. Calais, P.H., Pires, D.E.V., Guedes, D.O., Meira, W., Hoepers, C., Steding-jessen, K.: A campaign-based characterization of spamming strategies. In: Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS) (2008)
7. Egele, M., Kruegel, C., Kirda, E., Yin, H., Song, D.: Dynamic Spyware Analysis. In: Proceedings of the USENIX Annual Technical Conference. USENIX Association, Berkeley, CA, USA (2007)
8. Filiol, E.: Malware pattern scanning schemes secure against black-box analysis. Journal in Computer Virology 2(1), 35–50 (2006)
9. Filiol, E., Jacob, G., Liard, M.L.: Evaluation methodology and theoretical model for antiviral behavioural detection strategies. Journal in Computer Virology 3(1), 23–37 (2007)
10. Hoglund, G., Butler, J.: Rootkits—Subverting the Windows Kernel. Addison-Wesley (2006)
11. Jacob, G., Debar, H., Filiol, E.: Malware behavioral detection by attribute-automata using abstraction from platform and language. In: Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection. RAID ’09 (2009)

12. Kruegel, C., Kirda, E., Bayer, U.: TTAnalyze: A tool for analyzing malware. In: Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference (April 2006)
13. Kruegel, C., Kirda, E., Bayer, U., Balzarotti, D., Habibi, I.: Insights into current malware behavior. In: 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), Boston (April 2009)
14. Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., Mitchell, J.C.: A layered architecture for detecting malicious behaviors. In: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection. RAID '08 (2008)
15. Provos, N., Holz, T.: Virtual honeypots: from botnet tracking to intrusion detection. Addison-Wesley Professional, first edn. (2007)
16. SecureList: Rules for naming detected objects. `http://www.securelist.com/en/%20threats/detect?chapter=136`
17. Willems, C., Holz, T., Freiling, F.: Toward Automated Dynamic Malware Analysis Using CWSandbox. IEEE Security and Privacy 5, 32–39 (March 2007)