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

André Ricardo Abed Grégio[1] **, Paulo Lício de Geus**[2] **(Orientador) , Mario Jino**[3] **(Orientador)**

[1]CTI Renato Archer - MCT
[2]LAS - IC - Unicamp
[3]DCA - FEEC - Unicamp

argregio@cti.gov.br, paulo@las.ic.unicamp.br, jino@dca.fee.unicamp.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 article, we propose definitions to the different types of behavior that a program can present during its execution. Moreover, based on these behaviors, 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 extracted from a large set of actual malware samples and evaluate them in the context of these activities. Finally, we developed a system to translate from low-level execution traces to the proposed dangerous activities.

Keywords – Computer Security; Malware Analysis

1. Introduction

Malicious software is a major threat to Internet-connected systems. This kind of software ranges from worms and trojan horses to rootkits and bots and is generically referred to as malware. Thousands of malware variants are created periodically, hindering their analysis and the creation of vaccines by antiviruses companies. Moreover, publicly available dynamic analysis systems such as Anubis [5], CWSandbox [7], Norman [1] and ThreatExpert [2] provide reports filled with too many technical details and/or too much information in a slew of activities that can confuse the user on judging the malignity of an analyzed sample.

We propose a more generalized and abstract approach to describe malicious activities based on high-level behavior observed on 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 characterize 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.

Additionally, we have tested our proposed behavioral filters on a large set of actual malware samples that were gathered from malware collection honeypots [6] and spam attachments and then executed in our analysis system. 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 types.

2. Behavioral Traces

To the extent of this work, we use behavioral traces to pinpoint the main activities performed by a program. In this section we explain the procedure used to extract a behavioral trace so as to identify malicious activities and also define different forms of "behavior" in the scope of this article.

2.1. Data Processing

The first step to extracting a behavioral trace from a malware sample is to run it in 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 tool that captures selected system calls through SSDT hooking [4] on an emulated system (the guest) and registers them onto a file on a base system (the host).

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 an action's performer, the **operation**— i.e., the type of interaction, like

CreateProcess, DeleteFile and ConnectNetwork—between the source and the **target** of that operation.

The advantage of processing system calls in this 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 parametersBy abstracting from the particular variant chosen by the monitored program, we are able to present a much more meaningful result, that is, a `DeleteFile` operation.

2.2. 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 an 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 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**, whose actions can be active or passive, but that 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.

2.3. Suspicious Behavior

When a malicious program is executed, every action that is performed 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 connect to a machine, to write the file containing the malicious code on the compromised system and to create the process of the downloaded file that will handle the spreading process.

Thus, we are only interested in actions that modify the state of the compromised system (the active subset of the behavior, that is, B_A) and we want to avoid the actions that are considered normal to a program's execution (the neutral behavior, that is, B_N). To this end, 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 extracted $B_S(M_k)$, we derive a set of network and system-level actions that represent dangerous activities as regards the security of a system.

3. 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; they 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.

3.1. 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 username. An attacker can make use of this information to choose targets for an attack, or to map his 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.

Scanning. To find possible targets to spread to, a worm needs to perform a scan over the network. This involves the search of known vulnerable services or unprotected/open network applications.

DoS. There are classes of malware, such as bots, whose malicious 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 kinds 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 antivirus or other security mechanisms.

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.

3.2. System-level Risky Activities

Name Resolution File Modification. A trojan 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 (account number and password, credit card number). This modification allows a kind of malware commonly known as "banker" to have a user access a fake online banking site, for example.

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. On the other hand, a malware sample that is able to identify that it is being analyzed can also remove itself from the system.

Security Mechanisms Corruption. To ease compromising a target system, malware authors usually try to identify and terminate security mechanisms. This activity can be accomplished by turning off the system firewall or known antivirus engines, through the termination of their processes and removal of related registry keys.

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.

4. Experimental Results

We collected 1,641 malware samples from July, 2010 to July, 2011—463 from honeypots (**collector**) and 1,182 from spam (**phishing**)—and executed them in our dynamic analysis environment, which is a Qemu-emulated [3] MS Windows XP SP3, producing a behavior trace for each sample. We then applied the behavioral filters described in Section 3. to the traces and analyzed the network- and system-level malicious activities. In Section 4.1., we discuss the malicious activities observed over the full malware set.

4.1. 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 `FirewallPolicy\StandardProfile` registry keys, setting a "0" as the value of the **EnableFirewall** parameter.

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 lists of dangerous (and potentially malicious) activities attributed to our malware dataset. This process applied to the complete dataset produced the results from Table 1, divided by source (phishing or collector).

Table 1. Malicious activities discovered through the pinpointing process of our collection (P = Phishing and C = Collector); sum may be higher than 100%.

Level	Activity	P (%)	C (%)
Network	Evasion	3.72	6.69
	Scan	14.21	50.54
	DoS	37.22	29.37
	Download	1.10	9.07
	E-mail	1.95	3.45
	Hosts File	1.10	0.43
	Evidence	15.06	4.32
	Security Bypass	4.48	5.18
	Driver	5.16	0

We notice that most of the analyzed malware samples perform the same set of malicious activities, in disregard for their source. Those activities are attempts to scan networks for vulnerable services or UDP flooding at the network-level and self-removal and security mechanism bypass at the system-level. From the samples that came from our collectors, 15.15% did not present any behavior, either due to crashing on execution, to corrupted binaries or to analysis evasion. 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. Conclusion

In this paper, we proposed to define malware through their execution behavior. Our definition of suspicious behavior denotes dangerous activities performed at the network and system-level and encompasses only those that cause changes to the target system's state. We leveraged intelligible activities whose main goals are to help with incident response, to increase the understanding of malware analysis reports and to allow the behavioral classification of malware samples. To validate our approach, we tested a dataset composed by malware actually seen in the wild and collected from different sources. To this end, we built behavior filters that translated system calls to malicious activities, which were attached as an intermediate layer in our dynamic analysis system in order to bridge the semantic gap. As for the future, we are working on a classification process based on the behavioral filters we defined in this work, as well as on more precise definitions of risky activities to enhance the ability to do a better classification.

References

[1] Norman Sandbox. `http://www.norman.com/security_center/security_tools/`.

[2] ThreatExpert. `http://www.threatexpert.com/`.

[3] Fabrice Bellard. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*, pages 41–46, 2005.

[4] Greg Hoglund and James Butler. *Rootkits—Subverting the Windows Kernel.* Addison-Wesley, 2006.

[5] Christopher Kruegel, Engin Kirda, and Ulrich Bayer. TTAnalyze: A tool for analyzing malware. In *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, April 2006.

[6] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection.* Addison-Wesley Professional, first edition, 2007.

[7] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy*, 5:32–39, March 2007.