Monitoring User Actions for Better Malware Specifications

Peter Chapman
University of Virginia
pmc8p@virginia.edu

Jeffrey Shirley
University of Virginia
jshirley@cs.virginia.edu

***Abstract*—Malware authors have used the open nature of the web to distribute malicious software more quickly than security software manufacturers can respond using static signature-based detection mechanisms. Current behavioral techniques suffer from high false positive rates because of the difficulty of distinguishing between benign and malicious behaviors based only on system calls. We propose incorporating user actions to improve the precision of malware specifications and introduce a system to create effective application security policies based on the relationships between user interaction, GUI events, and run-time operations of both benign and malicious applications. Initial results show that GUI events provide promising additional information for distinguishing malicious and benign behavior.**

***Keywords*-security; malware detection; gui; user interaction;**

I. Introduction

Malware authors use the Internet to distribute new exploits and malicious applications more quickly and in higher volume than static signature-based detection techniques can feasibly handle [1]. Furthermore, the widespread use of metamorphic malware renders static signature-based scanners largely ineffective. The application of specification mining to develop accurate definitions for malicious and benign behavior is promising but has yet to be proven effective at creating generalized descriptions [2]. Our research applies specification mining techniques to create security policies based on the characteristics that distinguish malicious and benign applications. In addition to the traditional measures of program activity, we integrate user interaction and graphical user interface (GUI) events into the specifications. Our goal is to create a set of security policies that accurately define benign behavior as a series of temporally related and dependent system calls.

Strong policies should be developed to detect discrepancies between underlying actions of a program and GUI events. Most malware forgoes the use of a GUI in order to avoid alerting the user to its presence [3]. Malware often modifies system folders and registry entries, but benign applications generally only perform such actions in conjunction with a graphical installer combined with user input. Also, a trusted application can be hijacked to perform undesired operations, but the relationship between the GUI and the underlying actions signals suspicious behavior. For example, PDFs exploiting `CVE-2009-0927` [4] against Adobe Reader perform a JavaScript buffer-overflow to gain access to the target system. Adobe Reader can be exploited to conduct malicious activities never associated with viewing PDFs, such as registering new system services, downloading and executing arbitrary files, and deleting unassociated registry entries. Even though these actions would be associated with GUI events, they can be distinguished from the normal application behavior.

II. Approach

We built a system to derive application security policies based on the information and insight gained from GUI events and program execution traces. The security policy is represented as a finite-state machine that generalizes benign behavior across programs. Graph vertices represent logged API calls (including relevant GUI events), and edges are created between temporally linked and logically dependent events. Figure 1 depicts an overview of our approach.

A. Trace Collection

To evaluate and learn effective policies we used a representative dataset of 3000 malicious samples provided by the Anubis [5] project, including a variety of viruses, spyware, adware, worms, and exploits (e.g., malformed PDF and HTML files). We used VirusTotal [6], a web service that runs submitted files through dozens of professional security applications, to identify and organize the samples.

To collect execution traces, we built a custom framework that runs samples in a virtual machine (VM), logging all events of interest to the host. There is no guarantee we will see all malicious behavior in a single execution but we chose a timeout of 30 minutes based on observations during testing. The tracing software was implemented, in part, using Microsoft Detours [7], a library for intercepting Win32 API calls. Detours allows our tool to log user-level function calls (file, network, GUI, and user interaction) on each process running in the VM. Tracking the direct interaction of samples with the operating system circumvents complications brought by code obfuscation techniques [8] [9].

We scripted automated testing of interactive programs using the Java-based open source T-Plan Robot [10] software and simulated typical use cases of popular and representative applications to gather benign data. Common tasks, such as program launches, file opens, and window manipulation

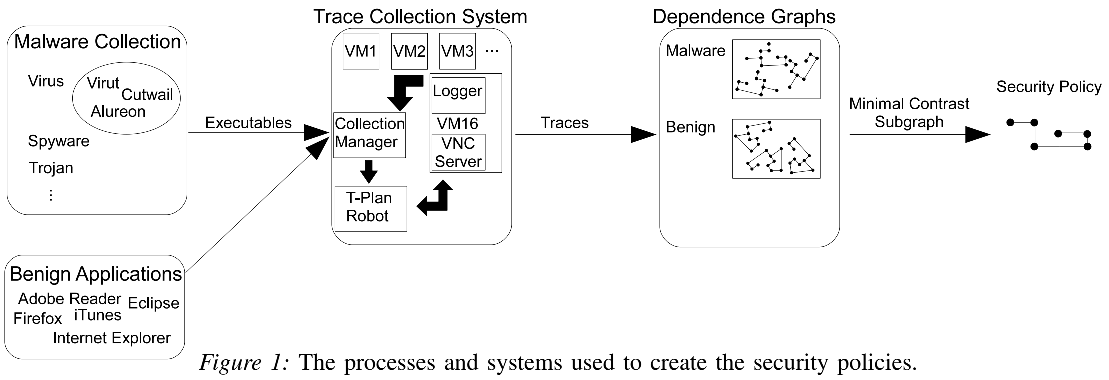

Figure 1: The processes and systems used to create the security policies.

are performed in a randomized order to limit experimental influence of the scripting. Some situations are inherently more difficult to evaluate as benign or malicious. Application installers are graphical and perform activities similar in many respects to malware. To compensate we included several program installations in our experiments.

B. Policy Creation

Based on the work of Christodorescu, Jha, and Kruegal [2], we used the collected traces to build dependence graphs for the malicious and benign applications. Each node in the graph is a system call recorded in the the execution trace, and edges are constructed between nodes that are data dependent (i.e., the output of a system call depends on a previous call) or occur within a specified time period, Δ. We add time-based edges in order to include GUI-initiated actions that do not directly share data with related system calls. For our preliminary experiments we used $\Delta = 2ms$.

After creating the dependency graphs we generalize them in order to create security policies that are not limited to the specific applications tested. We replace file paths and registry entries with generic variables (except for certain predefined key files such as system files, where the specific file is essential to the behavior). When a behavior flow is checked against the generated security policies, the abstract variables are bound to concrete ones.

The minimal contrast subgraph is a subgraph that appears in one set of graphs but not in another. We adapted the algorithm developed by Ting and Bailey [11] to find the minimal contrast subgraphs in our data. To minimize false negatives with an acceptable number of false positives, we find the contrast subgraphs for information flows present in the benign applications but not in the malware. The implemented algorithm is inefficient. We mitigate this limitation by only finding the contrast subgraphs for connected elements of the dependence graphs.

A sequence of events that correspond to an information flow in the security policy can be identified as benign. Sequences not found in the graph are to be treated with suspicion. For example, we hope to identify flows encapsulating the user interaction and GUI events related to creating a system file-chooser dialog followed by the creation of a file handle. Such a sequence is evident of user intentions.

III. Conclusion

We believe relations between the GUI and underlying actions provide insight into program behavior that can be valuable in the creation and enforcement of security policies. We have developed a system for automatically learning such policies from collected traces and are beginning to evaluate its effectiveness at detecting malware. By the symposium, we hope to be able to report on results from preliminary experiments.

References

[1] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," *Annual Computer Security Applications Conference*, pp. 421–430, 2007.

[2] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *ESEC-FSE '07*. New York, NY, USA: ACM, 2007, pp. 5–14.

[3] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," 2009.

[4] National Vulnerability Database. (2009) Vulnerability Summary for CVE-2009-0927. [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2009-0927

[5] International Secure Systems Lab. (2010) Anubis. [Online]. Available: http://anubis.iseclab.org/

[6] Hispasec Sistemas. (2010) VirusTotal. [Online]. Available: http://www.virustotal.com/

[7] R. Wa, G. Hunt, G. Hunt, D. Brubacher, and D. Brubacher, "Detours: Binary interception of Win32 functions," in *In Proceedings of the 3rd USENIX Windows NT Symposium*, 1998, pp. 135–143.

[8] M. Christodorescu and S. Jha, "Testing malware detectors," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 4, pp. 34–44, 2004.

[9] G. Jacob, "Behavioral detection of malware: from a survey towards an established taxonomy," *Journal in computer virology*, vol. 4, no. 3, pp. 251–266, 2008.

[10] T-Plan Limited. (2010) T-Plan Robot. [Online]. Available: http://www.vncrobot.com/

[11] R. M. H. Ting and J. Bailey, "Mining minimal contrast subgraph patterns," in *SDM*, 2006.