Poster: Automatic Profiling of Evasive Mixed-Mode Malware with SEMU

Shabnam Aboughadareh, Christoph Csallner
Computer Science & Eng Dept
University of Texas at Arlington
Arlington, TX 76019, USA
shabnam.aboughadareh@mavs.uta.edu
csallner@uta.edu

Mehdi Azarmi
Computer Science Dept
Purdue University
West Lafayette, IN 47907, USA
mazarmi@purdue.edu

I. Introduction

Malware and malware analysis are in an arms race. While analysis tries to understand the latest malware, malware tries to evade the latest analysis techniques. Modern malicious codes can evade analysis by manipulating structures that are important to the analysis, i.e., OS and virtual machine introspection components (VMI). An advanced evasion technique is orchestrating actions between user-mode and kernel-mode malware components. We also refer to such malware as *mixed-mode malware*. Such malware can evade many kinds of current dynamic malware analysis techniques, including those based on TEMU [5], Anubis [2], and Ether [3], such as dAnubis [4] and Panorama [6].

While many approaches have been proposed in the literature to analyze malware either at the user-level or at the kernel-level, mixed-mode malware has not received much research interest. However, this type of malware is very powerful and can be as dangerous as other forms of malware.

Specifically, previous dynamic malware analyses suffer from one or both of the following shortcomings. (1) Many current analysis techniques place analysis components in the same domain in which the malware is executing and thereby expose the analysis to malware manipulations. These approaches are referred to as *inside-the-box* or *inside-the-guest*. For example, popular analysis platforms such as TEMU and Anubis run the malware in a virtual machine. To inspect the state of the malware and the VM, such malware analyses often place some virtual machine introspection (VMI) components inside the VM, which exposes VMI and thereby the entire analysis to malware manipulation.

(2) Second, many approaches focus on a single domain, either kernel-mode or user-mode, but fail to fully capture malware that operates in both modes. For example, Ether leverages hardware extensions to operates outside-the-box but focuses only on user-mode analysis.

To address the shortcomings in the existing techniques we propose a novel dynamic malware analysis system called SEMU, which operates both outside-the-box and across kernel and user mode. SEMU leverages a reverse-engineered OS model to (a) capture the OS state before malware analysis, (b) track OS and malware activities, and (c) create an accurate log. SEMU can thus analyze many kinds of malware that evade existing techniques. In preliminary experiments we also found that SEMU's overhead was in between existing techniques, i.e., Ether and TEMU. While our current SEMU implementation is for Windows, our approach could also be implemented for other operating systems such as Linux.

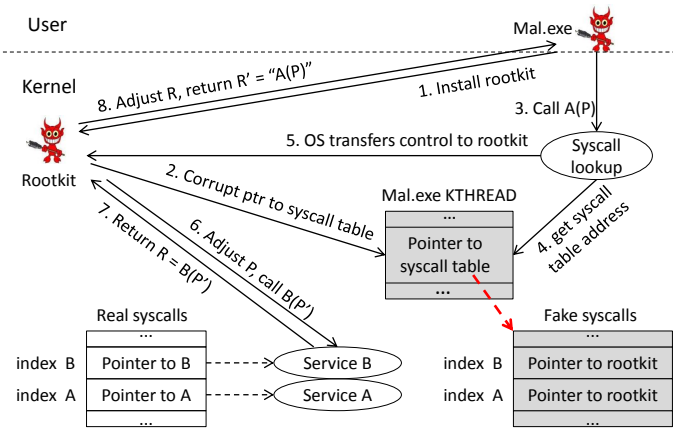

Fig. 1. Malware Mal.exe evades system call tracing and intrusion detection, by installing a rootkit that redirects service A system calls to service B; oval = original OS function; white box = original OS data; gray box = OS data created or manipulated by rootkit; solid arrow = attack; dashed arrow = target of original pointer; bold dashed arrow = target of manipulated pointer.

II. Motivating Example

Figure 1 shows an example attack that evades state-of-the-art outside-the-box malware analyses (such as Ether) that have shortcoming (2). The user-mode malware Mal.exe installs a kernel-mode component that redirects the subsequent system calls by Mal.exe, which breaks system call tracing. That is, Mal.exe can execute service B, which the analysis will log as a call to some benign service A.

While this example focuses on Ether-like single-mode analyses, it can be extended to also evade mixed-mode inside-the-guest analyses such as TEMU and Panorama. That is, the rootkit can detect the presence of inside-the-guest analysis components such as VMI and thereby evade the analysis.

III. Solution Overview

SEMU consists of four major components, (1) a reverse-engineered model of the guest OS, (2) a pre-analysis phase that captures the guest OS state before malware execution, (3) the malware monitoring phase, which creates a precise log of malware activities, and (4) a post-analysis log analysis.

Similar to TEMU, SEMU is implemented as an extension of a whole system emulator, QEMU. However, while TEMU places VMI components inside the guest OS, SEMU monitors the guest OS from outside the guest. SEMU infers the guest OS state via a carefully reverse-engineered model of the guest OS, derived from ReactOS[1] and Windows PDB symbols[2]. This model documents the address and layout of all kernel objects and functions.

Using this guest OS model, the pre-analysis phase identifies the location of trusted OS code and data. That is, this phase captures the clean OS state directly before malware execution. This state can differ between OS installations and executions and is therefore captured before each malware run.

During malware execution, SEMU logs key events in both user-mode and kernel-mode such as system calls, library calls, and IOCTLs in user-mode. In kernel-mode, SEMU tracks the execution of kernel functions that load new code (i.e., drivers) or create, modify, or delete objects. SEMU uses this information to update its model of the clean OS state, to maintain the address ranges of all current trusted OS objects and functions. This model allows SEMU to flag instruction executions as belonging to a rootkit. SEMU also logs all kernel-level malware operations. In the Figure 1 example, SEMU logs any malware system calls from user-mode in step 1, operations performed by the rootkit in steps 1 and 2, the malware's service A system call in step 3, and the rootkit's execution of service B in step 6.

The post-analysis aggregates the collected log, e.g., matching system calls from user-mode with operations invoked by kernel-mode malware. In the Figure 1 example, this post-processing matches the A call with the invocation of B, which reveals the malware's system call redirections.

IV. Preliminary Results

To evaluate our approach, we ask two research questions. Is the SEMU execution time competitive when compared with the most closely related approaches that are publicly available, i.e., TEMU and Ether (RQ1)? Can SEMU detect evasive mixed-mode malware that evades TEMU and Ether (RQ2)?

For RQ1, we compared the total SEMU execution time with TEMU (Table I) and Ether (Table II). SEMU was faster than TEMU as SEMU uses a newer QEMU version. SEMU was slower than Ether as Ether uses hardware extensions. But SEMU also works without these extensions.

Table I shows a coarse-grained VMI task, symbol extraction, which reports newly loaded images and periodically reports all currently running modules. We used an Ubuntu 11.04 host OS on a 1.6 GHz i7 machine with 8 GB RAM with a 512 MB RAM 32 bit Windows XP SP3 guest OS. SEMU uses QEMU v0.14 vs. v0.9 in TEMU. SEMU had both an overall lower runtime and a lower relative VMI overhead.

[1]http://www.reactos.org/

[2]http://msdn.microsoft.com/en-us/library/ff550665.aspx

Subject	w/o VMI [s]		coarse VMI [s]		VMI adds [%]	
	Temu	Semu	Temu	Semu	Temu	Semu
PsGetsid	1.68	0.56	3.44	1.09	105	95
Pslist -t	3.19	1.03	4.69	1.31	47	27
Psinfo -s	5.76	2.88	9.79	4.78	70	66
Coreinfo	1.70	0.65	3.75	1.07	121	63
LDLL -d ntdll.dll	3.20	2.58	5.01	3.75	57	45

TABLE I
Coarse-grained VMI: Symbol extraction in TEMU and SEMU; LDLL = ListDLLs.

Subject	w/o VMI [s]		fine VMI [s]		VMI slowdown	
	Ether	Semu	Ether	Semu	Ether	Semu
Efsinfo	0.63	2.42	20.54	21.39	32	8
Timezone /g	0.05	0.79	4.41	13.03	87	16
Whoami	0.03	0.72	4.49	19.83	149	27
UPX	0.32	9.00	45.58	322.60	141	35
RAR a	0.15	3.07	45.16	302.93	300	98

TABLE II
Fine-grained VMI: Instruction tracing in Ether and SEMU.

Table II shows a fine-grained VMI task—logging each instruction. We used a Debian Lenny domain-0 OS on a 2.33 GHz Xeon machine with 32 GB RAM with a 1 GB RAM 32 bit Windows XP SP2 guest OS. Overall, Ether was faster due to its use of hardware virtualization. However, instruction-level tracing slows Ether down, as it requires a debug exception after each instruction.

For RQ2, we developed and analyzed on TEMU, Ether, and SEMU several mixed-mode malware samples, e.g., Figure 1. The samples perform attacks including DKOM, DKSM [1], and hooking, by manipulating OS objects or data structures such as KTHREAD, EPROCESS, DRIVER_OBJECT, and SSDT. In this comparison SEMU was the only tool that could log all the events that are necessary for analyzing these attacks.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. 1017305 and 1117369.

References

[1] S. Bahram *et al.*, "DKSM: Subverting virtual machine introspection for fun and profit," in *Proc. 29th SRDS*. IEEE, Oct. 2010, pp. 82–91.

[2] U. Bayer, A. Moser, C. Krügel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, Aug. 2006.

[3] A. Dinaburg, P. Royal, M. I. Sharif, and W. Lee, "Ether: Malware analysis via hardware virtualization extensions," in *Proc. 15th CCS*. ACM, Oct. 2008, pp. 51–62.

[4] M. Neugschwandtner, C. Platzer, P. Comparetti, and U. Bayer, "dAnubis - dynamic device driver analysis based on virtual machine introspection," in *Proc. 7th DIMVA*. Springer, Jul. 2010, pp. 41–60.

[5] D. X. Song *et al.*, "BitBlaze: A new approach to computer security via binary analysis," in *Proc. 4th ICISS*. Springer, Dec. 2008, pp. 1–25.

[6] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing system-wide information flow for malware detection and analysis," in *Proc. 14th CCS*. ACM, Oct. 2007, pp. 116–127.