Dynamic Analysis of Evasive Modular Malware

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

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

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

1. INTRODUCTION AND MOTIVATION

Dynamic malware analysis tools usually rely on the integrity of kernel objects (data structures) for extracting the user-mode malware's behavior. On the other hand, malware developers can exploit this assumption and equip their malicious codes with kernel-level modules (rootkits) to compromise critical kernel objecs and thereby derail the analysis. The rootkit module of such a malware may prevent its user-mode component from being analyzed by state of the art malware analysis tools such as TEMU or Ether [1]. For instance, a rootkit can swap the `PID` value of a process object, belonged to its user-mode module, with a value for a benign process object and mislead a malware analysis tool that uses `PID` as an identifier for distinguishing the malware process.

Current approaches respond to such challenges by either giving up analysis or preventing the malware from manipulating the kernel mode data structures [2]. However this response is insufficient, as malware in turn can detect if its kernel manipulation attempts succeeded. If a manipulation attempt is not successful, malware may assume that it is being monitored by an analysis system and then proceed with a benign behavior to hide its malicious behavior.

We address the problem of evasive modular malware analysis with a novel shadow memory scheme that maintains a copy (shadow) of key kernel objects at runtime. All applications except malware would use the standard main memory. The copy (shadow memory) is designed to be used by malware. The advantage of this scheme is that a malware analysis system can permit malware to manipulate critical kernel data structures as these manipulations occur only in the shadow memory and therefore do not affect OS services or the malware analysis. This in turn tricks the malware into assuming it is not monitored and can perform its malicious behavior, which allows the malware analysis to observe the malicious behavior.

2. PROPOSED APPROACH

The key subsystem of our design is a virtual machine monitor (VMM) that maintains a copy of important kernel data structures and redirects each read or write operation performed by any piece of code to the appropriate memory version. Our current prototype keeps a set of fundamental data structures in shadow memory. The analyst can change that set to balance the level of protection with runtime overhead.

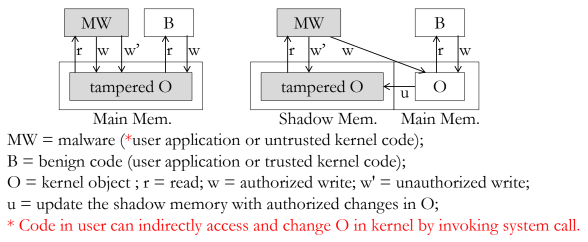
MW
B
r
w
w'
r
w
tampered O
Main Mem.
MW
B
r
w'
w
r
w
tampered O
u
O
Shadow Mem.
Main Mem.
MW = malware (*user application or untrusted kernel code);
B = benign code (user application or trusted kernel code);
O = kernel object ; r = read; w = authorized write; w' = unauthorized write;
u = update the shadow memory with authorized changes in O;
* Code in user can indirectly access and change O in kernel by invoking system call.

Figure 1: Result of tampering by malware in standard analysis (left) vs. in our scheme (right).

Figure 1 gives an overview of how the VMM redirects read and write operations of a set of OS objects/data structures **O** included in the scheme. (All reads and writes of other kernel objects are served from main memory.) At the core, the VMM detects from which process or kernel code an operation originates and which parts of a data structure it affects. The VMM directs all reads and writes of benign user-mode applications and trusted kernel code to the main memory. If a write originates from an untrusted kernel code and affects a critical part of **O**, the VMM directs that write to shadow memory. Other write operations (by malware, a benign application or kernel) to **O** objects are directed to main memory, but then written back to the shadow memory. The VMM serves all **O** reads by the malware from shadow memory.

Initial Results: Our prototype implementation on top of QEMU suggests that our design is able to analyze several malware samples that use DKSM attacks [1] and cannot be analyzed with TEMU or Ether, while our approach has a similar overhead.

3. REFERENCES

[1] S. Bahram, X. Jiang, Z. Wang, M. Grace, J. Li, D. Srinivasan, J. Rhee, and D. Xu. DKSM: Subverting virtual machine introspection for fun and profit. In *Proc. 29th IEEE Symposium on Reliable Distributed Systems (SRDS)*. IEEE, Oct. 2010.

[2] R. Riley, X. Jiang, and D. Xu. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing. In *Proc. 11th International Symposium on Recent Advances in Intrusion Detection (RAID)*. Springer, 2008.