System Call API Obfuscation (Extended Abstract)

Abhinav Srivastava[1], Andrea Lanzi[1,2], and Jonathon Giffin[1]

[1]School of Computer Science, Georgia Institute of Technology, USA
[2]Dipartimento di Informatica e Comunicazione, Università degli Studi di Milano, Italy
{abhinav,giffin}@cc.gatech.edu, andrew@security.dico.unimi.it

Abstract. We claim that attacks can evade the comprehension of security tools that rely on knowledge of standard system call interfaces to reason about process execution behavior. Our attack, called *Illusion*, will invoke privileged operations in a Windows or Linux kernel at the request of user-level processes without requiring those processes to call the actual system calls corresponding to the operations. The Illusion interface will hide system operations from user-, kernel-, and hypervisor-level monitors mediating the conventional system-call interface. Illusion will alter neither static kernel code nor read-only dispatch tables, remaining elusive from tools protecting kernel memory.

1 Illusion Attack

Honeypots and other utilities designed to audit, understand, classify, and detect malware and software attacks often monitor process' behavior at the system call interface as part of their approach. Past research has developed a widespread collection of system-call based systems operating at user or kernel level [1,5,2,4] and at hypervisor level [3]. Employing reference monitors at the system call interface makes intuitive sense: absent flaws in the operating system (OS) kernel, it a non-bypassable interface, so malicious code intending to unsafely alter the system will reveal its behavior through the series of system calls that it invokes.

Current malware increasingly makes use of kernel modules or drivers that help the user-level process perform malicious activities by hiding the process' side effects. For example, the rootkits *adore* and *knark* hide processes, network connections, and malicious files by illegitimately redirecting interrupt or system call handling into their kernel modules. Redirection can alter the semantic meaning of a system call—a problem for any system that monitors system calls to understand the behavior of malware. Jiang and Wang address this class of attack:

> Syscall remapping requires the modification of either the interrupt descriptor table (IDT) or the system call handler routine... [3]

Systems like that of Jiang and Wang assume that protections against illegitimate alteration of the IDT or system call handler will force malicious software to

R. Lippmann, E. Kirda, and A. Trachtenberg (Eds.): RAID 2008, LNCS 5230, pp. 421–422, 2008.

always follow the standard system-call interface when requesting service from the kernel.

Unfortunately, this assumption does not hold true. Malicious code can obfuscate the Windows or Linux system call interface using only legitimate functionality commonly used by kernel modules and drivers. Our *Illusion* attack will allow malicious processes to invoke privileged kernel operations without requiring the malware to call the actual system calls corresponding to those operations. In contrast to prior attacks of the sort considered by Jiang and Wang, Illusion will alter neither static kernel code nor read-only dispatch tables such as the IAT or system call descriptor table (SSDT). During the execution of malware augmented with the Illusion attack, an existing system-call analyzer will see a series of system calls different that those actually executed by the malware.

The Illusion attack is possible because a number of system calls allow *legitimate* dispatch into code contained in a kernel module or driver, and this permits an attacker to alter their semantics. Consider `ioctl`: this system call takes an arbitrary, uninterpreted memory buffer as an argument and passes that argument to a function in a kernel module that has registered itself as the handler for a special file. Benign kernel modules legitimately register handler functions for such files; a malicious module performing the same registration exhibits no behaviors different than the benign code. However, a call to `ioctl` will be directed into the malicious module's code together with the buffer passed to `ioctl` as an argument. In user-space, we marshal a malware's actual system call request into this buffer, and we then use the kernel module to unmarshal the request and invoke the appropriate kernel system call handler function. With this interface illusion in place, the kernel still executes the same operations that the malware instance would have executed without the obfuscation. However, system call monitoring utilities would observe a sequence of `ioctl` requests and would not realize that malicious operations had occurred.

References

1. Forrest, S., Hofmeyr, S.A., Somayaji, A., Longstaff, T.A.: A sense of self for UNIX processes. In: IEEE Symposium on Security and Privacy, Oakland, CA (May 1996)
2. Giffin, J.T., Jha, S., Miller, B.P.: Efficient context-sensitive intrusion detection. In: Network and Distributed System Security Symposium (NDSS), San Diego, CA (February 2004)
3. Jiang, X., Wang, X.: Out-of-the-box monitoring of VM-based high-interaction honeypots. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 198–218. Springer, Heidelberg (2007)
4. Krohn, M., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: Symposium on Operating System Principles (SOSP), Stevenson, WA (October 2007)
5. Sekar, R., Bendre, M., Dhurjati, D., Bollineni, P.: A fast automaton-based method for detecting anomalous program behaviors. In: IEEE Symposium on Security and Privacy, Oakland, CA (May 2001)