SPAD: Software Protection through Anti-debugging Using Hardware Virtualization

Qian Lin, Mingyuan Xia, Miao Yu, Peijie Yu, Min Zhu, Shang Gao, Zhengwei Qi, Kai Chen, Haibing Guan

Shanghai Jiao Tong University, Shanghai, P.R.China
{ linqian, kenmark, superymk, yupiwang, zhumin, chillygs, qizhwei, kchen, hbguan } @ sjtu.edu.cn

Abstract

Debugging could be a threat to system security when adopted by malicious attackers. The major challenges of software-only anti-debugging are compromised strategy and lack of self-protection. Leveraging hardware virtualization, we proposes a strategy of software protection through anti-debugging which imperceptibly monitors the debug event on a higher privilege level than the conventional kernel space. Our prototype can effectively prohibit the debugging behavior from selected popular debuggers in the replication experiment.

Categories and Subject Descriptors

D.4.4 [**Operating Systems**]: Security and Protection

General Terms

Design, Security

Keywords

Anti-debugging, Software protection, Virtualization

1. INTRODUCTION

The requirement for software protection has gained general attention in the digital world. A great variety of copy-protection strategies have been developed to prevent cracking, tracing and reverse engineering. The majority of these mechanisms provide a reasonable level of security against static-only analysis. Nevertheless, most of them are vulnerable to dynamic or hybrid static-dynamic attacks which are commonly used by hackers. Although debugging is usually employed in the software development to help find software bugs or deficiencies, it also facilitates hackers to reverse commercial software or steal private information [1]. Hackers could utilize a generic debugger to set a breakpoint at the password input function, scan the specific buffer and wait for the debugger window to reveal the real password. Similar scenarios may be applied to other private and sensitive account data acquirement. Some software may be packed and protected by packers to fight against reverse engineering, but most packers suffer from the compatibility problem, resulting in erroneous effects to the protected applications.

Virtual machine can be an effective environment to prevent software from attacks and malicious behavior [2]. Unlike conventional methods which rely on kernel space to construct anti-debugging architecture, we leverage hardware virtualization to occupy a higher privilege level so that not a single debugging behavior can escape from our protection shield. We implement the prototype of Software Protection through Anti-debugging (SPAD) and verify its effectiveness.

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

2. BACKGROUND

To be effective, dynamic program analysis requires the target to be executed with sufficient test inputs to produce meaningful behavior. We classify all prevailing debuggers into three categories based on the difference of debugging mechanisms. *User-mode debugger* deploys the API to play debugging tricks. *Kernel debugger* exploits kernel resources to rule the debugging procedure. *System-level debugger* runs underneath OS and exhibits a strong ability to suspend all target operations.

3. DESIGN AND IMPLEMENTATION

Leveraging hardware virtualization, SPAD intercepts debugging event in *root mode* where the guest OS could not detect the procedure. Thus the transparency to the guest OS is guaranteed and no modification is needed to the guest application. We tailored *BluePill* as the base of SPAD implementation. The thin hypervisor with hardware virtualization support only monitors the sensitive behavior instead of managing the whole guest OS.

3.1 Anti-debugging functionality

According to the debugging mechanism of Windows, a process switching will happen if a debugger takes over the target process within the debugging exception handler. By configuring VMM, we can intercept the sensitive behavior including INT3, INT1 exceptions, `CPUID` instruction and CR3 register refreshing. SPAD monitors critical functions by hooking their entry and exit, respectively. This is done by replacing the function entry with a trampoline, which executes `CPUID` instruction before the original entry. Figure 1 illustrates that the system-level debugger's behavior will be detected as soon as it triggers the INT3 exception. Both

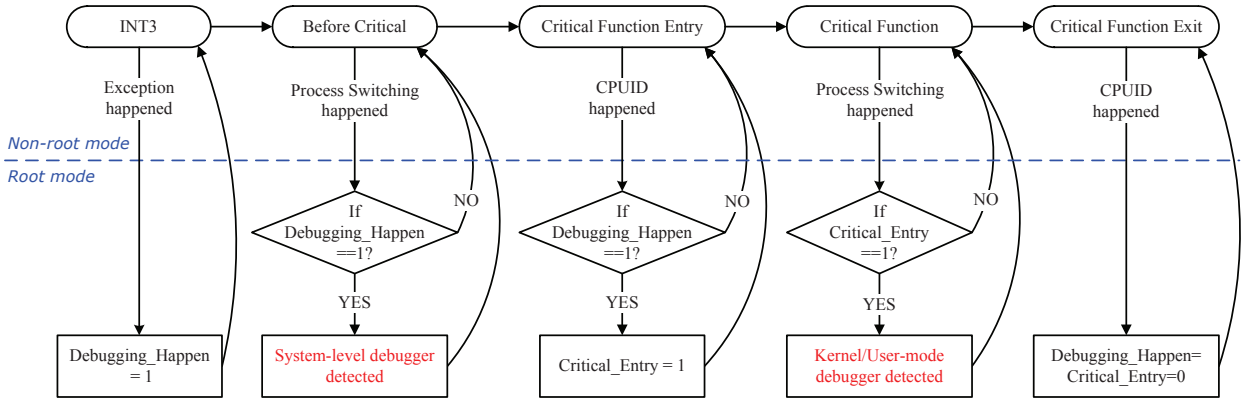

Figure 1: Anti-debugging Approach. Hypervisor monitors the sensitive events and triggers mode switching when certain events happen.

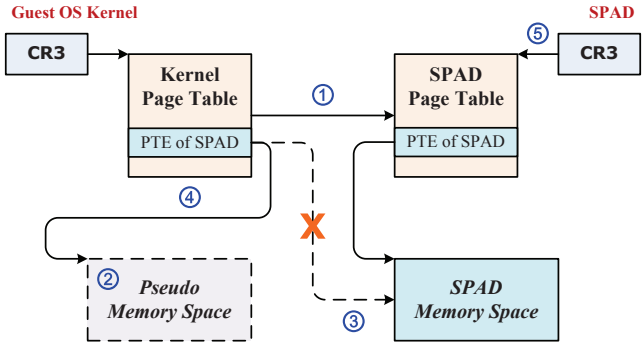

Figure 2: Construct Self-protection.

user-mode and kernel debuggers can be detected by checking whether process switching to the debugger happens in the critical function of *KiDispatchException()*.

3.2 Self-protection

If a malware or rootkit attempts to invade SPAD's memory space which is under the management of guest OS, it just tampers with SPAD's virtual memory space to deprive it of its functionality. Therefore, SPAD takes advantage of hardware virtualization and memory remapping techniques to conceal itself so that the threat of external attack could be eliminated. In order to implement self-protection, we modify the corresponding address mapping of guest OS's page table. Figure 2 depicts the steps of realizing this strategy, involving copying a new private page table for SPAD, creating a pseudo memory space and modifying the mapping of kernel page table. Consequently, VMM can use the private page table for its memory space access. Furthermore, the modification to the kernel page table is done in the *root mode*, which shields the operations from being detected by the guest OS.

4. EVALUATION

We choose 13 traditional anti-debugging methods to compare with SPAD against 9 kinds of generic debuggers. For user-mode debuggers, certain hiding plug-ins could facilitate debuggers to conceal themselves. However, since user-mode debuggers rely on APIs to communicate with system debugging server, it is highly possible to be detected by kernel modules. The result reveals that OllyDbg and OllyICE fail in more than half of the tests without hiding plug-ins. As for kernel debuggers, WinDbg features no anti anti-debugging techniques and fails at all anti-debugging methods. System-level debuggers pervade the operating system and attain better stealth property, whereas they leave more or less debugging interfaces in system and can be compromised once exploited.

In comparison with software-only anti-debugging methods, SPAD is more effective because it holds an even higher privilege level than the system-level debugger, and intercepts the sensitive system behavior instead of interacting with system components. Thus it can silently detect the debuggers while remaining risk-free from being detected by interfaces exposed to the system. Our experiments show that SPAD can detect all debuggers we selected.

5. CONCLUSION

This paper presents a lightweight and transparent protection strategy based on the art of virtualization. Leveraging hardware assisted virtual machine monitor, SPAD can detect and intercept debugging behavior in a higher privilege position than OS kernel's, as well as make itself imperceptible to debuggers.

6. ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China (Grant No.60773093, 60873209, 60970107), the Key Program for Basic Research of Shanghai (Grant No.09JC1407900, 09510701600, 10511500100), the Opening Project of Shanghai Key Lab of Advanced Manufacturing Environment (No.KF200902), IBM SUR Funding and IBM Research-China JP Funding, and supported by Key Lab of Information Network Security, Ministry of Public Security.

7. REFERENCES

[1] M. N. Gagnon, S. Taylor, and A. K. Ghosh. Software protection through anti-debugging. *IEEE Security & Privacy*, 5(3):82–84, 2007.

[2] S. T. King and S. W. Smith. Virtualization and security: Back to the future. *IEEE Security & Privacy*, 6(5):15, 2008.