Practical Timing Side Channel Attacks Against Kernel Space ASLR

Ralf Hund, Carsten Willems, and Thorsten Holz
Horst-Goertz Institute for IT Security (HGI)
Ruhr-University Bochum, Germany

Abstract

Due to the prevalence of control-flow hijacking attacks, a wide variety of defense methods to protect both user space and kernel space code have been developed. A few examples that have received widespread adoption include stack canaries, non-executable memory, and Address Space Layout Randomization (ASLR). When implemented correctly (i.e., a given system fully supports these protection methods and no information leak exists), typical exploitation strategies are severely thwarted.

In this short paper, we study the limitations of kernel space ASLR against a local attacker with restricted privileges. We show that an adversary can implement a generic *side channel attack against the memory management system to deduce information about the privileged address space layout. Our approach is based on the intrinsic property that the different caches are shared resources on computer systems. We introduce two implementations of our methodology and show that our attacks are feasible on four different x86-based CPUs (both 32- and 64-bit architectures) and also applicable to virtual machines. As a result, we can successfully circumvent kernel space ASLR on systems both running Windows and Linux.*

1 Introduction

Modern operating systems employ a wide variety of methods to protect both user and kernel space code against memory corruption attacks that leverage vulnerabilities such as stack overflows, integer overflows, and heap overflows. These control-flow hijacking attempts pose a significant threat and have attracted a lot of attention in the security community due to their high relevance in practice. To thwart such attacks, many mitigation techniques have been developed over the years. A few examples that have received widespread adoption include stack canaries [5], non-executable memory [10], and Address Space Layout Randomization (ASLR) [2, 12, 17]. Especially ASLR plays an important role in protecting computer systems against software faults. The key idea behind this technique is to randomize the system's virtual memory layout either every time a new code execution starts or every time the system is booted. While the initial implementations focussed on randomizing user mode processes, modern operating systems randomize *both* user and kernel space. All major operating systems such as Windows, Linux, and Mac OS X have adopted ASLR and also mobile operating systems like Android and iOS have recently added support for ASLR.

In combination with DEP, a technique that enforces the $W \oplus X$ (**W**ritable **xor** e**X**ecutable) property of memory pages, ASLR significantly reduces the attack surface. Under the assumption that the randomization itself cannot be predicted due to implementation flaws (i.e., not fully randomizing the system or existing information leaks), typical exploitation strategies are severely thwarted.

In this paper, we study the limitations of kernel space ASLR against a local attacker with restricted privileges. We introduce a generic attack for systems running on the Intel *Instruction Set Architecture* (ISA). More specifically, we show how a local attacker with restricted rights can mount a timing-based side channel attack against the memory management system to deduce information about the privileged address space layout. We take advantage of the fact that the memory hierarchy present in computer systems leads to shared resources between user and kernel space code that can be abused to construct a side channel. In practice, timing attacks against a modern CPU are very complicated due to the many performance optimizations used by current processors such as hardware prefetching, speculative execution, multi-core architectures, or branch prediction that significantly complicate timing measurements [11]. Previous work on side-channels attacks against CPUs [1, 6, 16] focused on older processors without such optimization and we had to overcome many challenges to solve the intrinsic problems related to modern CPU features [11].

We have implemented two attack strategies that are capable of successfully reconstructing (parts of) the kernel memory layout. We have tested these attacks on different Intel and AMD CPUs (both 32- and 64-bit architectures) on machines running either Windows 7 or Linux. Furthermore,

we show that our methodology also applies to virtual machines. As a result, an adversary learns precise information about the (randomized) memory layout of the kernel. With that knowledge, she is able to perform control-flow hijacking attacks to escalate her privileges since she now knows where to divert the control flow to, thus overcoming the protection mechanisms introduced by kernel space ASLR.

An extended version with implementation details and more attacks is available as a full paper [8].

2 Timing Side Channel Attacks

We focus in this work on local attacks against kernel space ASLR: we assume an adversary who already has restricted access to the system, i.e., she can run arbitrary applications, but does not have access to privileged kernel components and thus cannot execute privileged (kernel mode) code. We also assume the presence of a user mode-exploitable vulnerability within kernel or driver code, a common problem [4]. The exploitation of this software fault requires to know (at least portions of) the kernel space layout since the exploit at some point either jumps to an attacker controlled location or writes to an attacker controlled location to divert the control flow. Furthermore, we assume that the system correctly implements ASLR (i.e., the complete system is randomized and no information leaks exist) and that it enforces the $W \oplus X$ property. Hence, all typical exploitation strategies are thwarted.

Side channels emerge from intricacies of the underlying hardware and the fact that parts of the hardware (such as caches and physical memory) are *shared* between both privileged and non-privileged code. Note that all the approaches are applicable to many operating systems: while we tested our approach mainly on Windows 7 and Linux, we are confident that the attacks also apply for other versions of Windows or even other operating systems. Furthermore, our attacks apply to both 32- and 64-bit systems.

The methodology behind our timing measurements is as follows: At first, we attempt to set the system in a specific state from user mode. Then we measure the duration of a certain memory access operation. The time span of this operation then (possibly) reveals certain information about the kernel space layout. Our timing side channel attacks can be split into two categories:

- **L1/L2/L3-based Tests:** These tests focus on the L1/L2/L3 CPU caches and the time needed for fetching data and code from memory.
- **TLB-based Tests:** These tests focus on TLB and paging structure caches and the time needed for address translation.

To illustrate the approach, consider the following example: we make sure that a privileged code portion (such as the operating system's system call handler) is present within the caches by executing a system call. Then, we access a designated set of user space addresses and execute the system call again. If the system call takes longer than expected, then the access of user space addresses has evicted the system call handler code from the caches. Due to the structure of modern CPU caches, this reveals parts of the physical (and possibly virtual) address of the system call handler code as we show in our experiments.

3 Implementation and Results

We now briefly sketch two different implementations of timing side channel attacks. The goal of each attack is to precisely locate some of the currently loaded kernel modules from user mode by measuring the time needed for certain memory accesses. Note that an attacker can already perform a ROP-based attack once she has de-randomized the location of a few kernel modules or the kernel [7, 14].

We have evaluated our implementation on the 32-bit and 64-bit versions of *Windows 7 Enterprise* and *Ubuntu Desktop 11.10* on the following (native and virtual) hardware architectures to ensure that they are commonly applicable on a variety of platforms:

1. Intel i7-870 (Nehalem/Bloomfield, Quad-Core)
2. Intel i7-950 (Nehalem/Lynnfield, Quad-Core)
3. Intel i7-2600 (Sandybridge, Quad-Core)
4. AMD Athlon II X3 455 (Triple-Core)
5. VMWare Player 4.0.2 on Intel i7-870 (with VT-x)

Table 1 provides a high-level overview of our methods, their requirements, and the obtained results.

3.1 First Attack: Cache Probing

Our first method is based on the fact that multiple memory addresses have to be mapped into the same cache set and, thus, compete for available slots. This can be utilized to infer (parts of) virtual or physical addresses indirectly by trying to evict them from the caches in a controlled manner. More specifically, our method is based on the following steps: first, the searched code or data is loaded into the cache indirectly (e.g., by issuing an interrupt or calling `sysenter`). Then certain parts of the cache are consecutively replaced by accessing corresponding addresses from a user-controlled *eviction buffer*, for which the addresses are known. After each replacement, the access time to the searched kernel address is measured, for example by issuing the system call again. Once the measured time is significantly higher, one can be sure that the previously accessed eviction addresses were mapped into the same cache set. Since the addresses of these *colliding locations* are known, the corresponding cache index can be obtained and obviously this is also a part of the searched address.

Method	Requirements	Results	Environment	Success
Cache Probing	*large pages* or physical address of *eviction buffer*	`ntoskrnl.exe` and `hal.sys`	all	✓
Double Page Fault	TLB entry creation on access violation	allocation map, several drivers	all but AMD	✓

Table 1. Summary of timing side channel attacks against kernel space ASLR.

3.2 Second Attack: Double Page Fault

The second attack allows us to reconstruct the allocation map of the entire kernel space from user mode. To achieve this goal, we take advantage of the behavior of the TLB cache on Intel CPUs and within the virtual machine. The TLB typically works in the following way: whenever a memory access results in a successful page walk due to a TLB miss, the MMU replaces an existing TLB entry with the result of the page walk. Accesses to non-allocated virtual pages (i.e., the present bit in the PDE or PTE is set to zero) induce a page fault and the MMU does *not* create a TLB entry, since future accesses to the same page will always produce a page fault which is costly anyway. However, when the page translation was successful, but the access permission check fails (e.g., when kernel space is accessed from user mode), a TLB entry is indeed created.

This behavior can be exploited to reconstruct the entire kernel space from user mode by accessing each kernel space page p twice and measuring the access time. Due to performance optimizations of modern CPUs and the concurrency related to multiple cores, a single measurement can contain noise and outliers. We thus probe the kernel space multiple times and only use the observed minimal access time for each page to reduce measurement inaccuracies.

4 Conclusion

In this short paper, we have discussed a generic, timing-based side channel attack against kernel space ASLR. We apply the basic principle behind side channel attacks (e.g., [3, 6, 9, 13, 15]) and introduced different ways how this methodology can be used to obtain information about the memory layout of a given system. To the best of our knowledge, we are the first to demonstrate timing attacks against ASLR implementations.

More information about our work and implementation details are available in a full paper [8]. In this extended version, we also discuss a third attack against kernel space ASLR.

References

[1] O. Aciiçmez, B. B. Brumley, and P. Grabher. New Results on Instruction Cache Attacks. In *Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, 2010.

[2] S. Bhatkar, D. C. DuVarney, and R. Sekar. Address Obfuscation: An Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *USENIX Security Symposium*, 2003.

[3] D. Brumley and D. Boneh. Remote Timing Attacks are Practical. In *USENIX Security Symposium*, 2003.

[4] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. An Empirical Study of Operating Systems Errors. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2001.

[5] C. Cowan, C. Pu, D. Maier, H. Hintony, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, and Q. Zhang. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *USENIX Security Symposium*, 1998.

[6] D. Gullasch, E. Bangerter, and S. Krenn. Cache Games – Bringing Access-Based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy*, 2011.

[7] R. Hund, T. Holz, and F. C. Freiling. Return-Oriented Rootkits: Bypassing Kernel Code Integrity Protection Mechanisms. In *USENIX Security Symposium*, 2009.

[8] R. Hund, C. Willems, and T. Holz. Practical Timing Side Channel Attacks Against Kernel Space ASLR. In *IEEE Symposium on Security and Privacy*, 2013.

[9] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *International Crytology Conference (CRYPTO)*, 1996.

[10] Microsoft. Data Execution Prevention (DEP). `http://support.microsoft.com/kb/875352/EN-US/`, 2006.

[11] K. Mowery, S. Keelveedhi, and H. Shacham. Are AES x86 Cache Timing Attacks Still Feasible? In *ACM Cloud Computing Security Workshop (CCSW)*, 2012.

[12] PaX Team. Address Space Layout Randomization (ASLR). `http://pax.grsecurity.net/docs/aslr.txt`.

[13] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *ACM Conference on Computer and Communications Security (CCS)*, 2009.

[14] E. J. Schwartz, T. Avgerinos, and D. Brumley. Q: Exploit hardening made easy. In *USENIX Security Symposium*, 2011.

[15] D. X. Song, D. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *USENIX Security Symposium*, 2001.

[16] E. Tromer, D. A. Osvik, and A. Shamir. Efficient Cache Attacks on AES, and Countermeasures. *J. Cryptol.*, 23(2), Jan. 2010.

[17] J. Xu, Z. Kalbarczyk, and R. K. Iyer. Transparent Runtime Randomization for Security. In *Symposium on Reliable Distributed Systems (SRDS)*, 2003.