THE INTEL SYSRET PRIVILEGE ESCALATION

By George Dunlap | June 13, 2012 | Uncategorized

The Xen Security team recently disclosed a vulnerability, Xen Security Advisory 7 (CVE-2012-0217), which would allow guest administrators to escalate to hypervisor-level privileges. The impact is much wider than Xen; many other operating systems seem to have the same vulnerability, including NetBSD, FreeBSD, some versions of Microsoft Windows (including Windows 7).
So what was the vulnerability? It has to do with a subtle difference in the way in which Intel processors implement error handling in their version of AMD's `SYSRET` instruction. The `SYSRET` instruction is part of the x86-64 standard defined by AMD. If an operating system is written according to AMD's spec, but run on Intel hardware, the difference in implementation can be exploited by an attacker to write to arbitrary addresses in the operating system's memory. This blog will explore the technical details of the vulnerability.

It should be noted that there are some questions about the way in which the disclosure process for this vulnerability was handled. In order to make sure that everyone can fully participate in the discussion, including system administrators who may at this moment be patching their systems, there has been a conscious decision to postpone any discussion about the disclosure process for one week, until the 19th of June.

INTRODUCTION: CANONICAL ADDRESSES, CONTEXT SWITCH, AND SYSRET

The first concept to understand is that of *canonical addresses*. When designing the 64-bit extensions to x86, AMD didn't extend the actual virtual address space to a full 64 bits, but only to 48 bits. (This was mainly done to balance the benefits of extended address spaces with the cost of extra levels of the page table: 64 bits of virtual address space would have required a 6-level pagetable; 48 bits gives you 256 terabytes of virtual address space, far more than the largest machines will have for many years to come, and only requires 4 levels of pagetable.)

They could have made the processor just ignore the extra 16 bits of address space, but clever programmers have a tendency to take advantage of these kinds of quirks to do things like storing extra information in the "unused" portions of virtual addresses. These clever tricks would break, however, if the processors ever decided to extend the address space into more bits. In order to prevent this, they put in restrictions to the microprocessor: Any 64-bit value that is used as a virtual address (for instance, written into the instruction pointer register, `RIP`) must be in *canonical form*: That is, bits 48-63 must be the same as bit 47. Valid (or "canonical") addresses space thus exists in two non-contiguous 128-terabyte chunks: from `0x0000000000000000-0x00007fffffffffff`, and then from `0xffff800000000000-0xffffffffffffffff`. Any time a 64-bit value is used as a virtual address that does not fit in one of these two ranges, the processor will throw a general protection fault (`#GP`).

The next thing to understand is the process of context switching between the guest and the hypervisor. Whenever switching from a lower privilege to a higher privelege (either a kernel or a hypervisor), the registers of the lower-level privilege must be saved and replaced with those of the higher privilege. When an interrupt or exception is delivered while in guest mode, the `RIP`, stack pointer, and code segment (which encodes the privilege level) are changed by the hardware; the hypervisor then stores the rest of the guest state on its stack, and handles the interrupt.

In the early days of x86, a system call was treated as a special-case of an interrupt or exception: typically the guest would execute INT instruction to enter the hypervisor or operating system, and the hypervisor would execute IRET to return to the guest. But both AMD and Intel's performance analysis teams determined that for processes which made a significant number of system calls, going through the normal exception path in the processor was significantly slower. So they independently came up with special instructions for voluntarily escalating privilege: AMD came up with `SYSCALL` / `SYSRET`, and Intel came up with `SYSENTER` / `SYSEXIT`. They have slightly different semantics, and they're not available in all modes on both processors. Since `SYSRET` is part of the x86-64 standard, and available on all processors in 64-bit mode, it's common for 64-bit operating systems and hypervisors to use it exclusively.

Intel's version of SYSRET, however, has a very subtle difference in behavior than AMD's version. It is this difference that is the source of the problem.

SYSRET ON INTEL AND AMD

The key difference between `SYSCALL` / `SYSRET` and `INT` / `IRET` is how much information is stored and where. `SYSCALL` / `SYSRET` instructions only save and restore the `RIP` of the guest, as well as changing the code segment selector (which effectively changes the privilege level). Furthermore, rather than reading IDTs and writing to stack frames, the instructions copy the guest `RIP` to and from the `RCX` register, and the segment selectors and hypervisor `RIP` to and from various other processor registers. The hypervisor is responsible for saving and restoring all of the other registers, including the stack pointer (

`RSP`).
So the end-to-end effect of the `SYSRET` instruction, on both Intel and AMD, is as follows:

- Load `RIP` from `RCX`
- Change code segment selector to guest mode

Now, as we said before, `RIP` is used as a virtual address, so it must be canonical. `RCX`, however, is a general purpose register, and may contain any 64-bit number. So if for some reason there is a non-canonical address in `RCX`, upon executing the `SYSRET` instruction, the processor will throw a general protection fault (`#GP`).
Now here comes the subtle difference. The AMD and Intel architecture specifications include pseudocode that is meant to be a precise description of what the instruction does. In the AMD pseudocode, there is no explicit mention of the check for a canonical address; however, the actual `RIP` is not assigned until after the privilege level has been changed back to guest mode; and experimentation has confirmed that a `SYSRET` executed on AMD with a non-canonical address in `RCX` will throw the `#GP` in guest mode. However, in the Intel pseudocode, the check for a guest canonical address is explicit, and happens before the privilege level is changed. This means that if a `SYSRET` is executed on an Intel processor with a non-canonical address in `RCX`, the processor will throw a `#GP` in priveleged mode.
It's a subtle difference, but it's important, because the value of the stack pointer used when the processor delivers a `#GP` will change depending on the privilege level. If a `#GP` is delivered in guest mode, the processor will load the hypervisor's `RSP` from a special hypervisor-designated entry point. But if the `#GP` is delivered while in hypervisor mode, the processor will use the current `RSP`, so that it can effectively "nest" the exception.
But recall that the `SYSRET` instruction doesn't restore the `RSP`; the hypervisor has to do that. So by the time the `#GP` happens, the hypervisor has already restored the `RSP` to what was in the *guest's* `RSP` when it did the `SYSCALL`, along with all of the other general-purpose registers.
So if the guest can induce the hypervisor to have a non-canonical `RIP` to return to, it can set the `RSP` to any value it wants to in hypervisor memory, and get the hardware (in delivering a `#GP`) to overwrite it. This can in turn be leveraged into a full exploit (the details of which are left as an exercise to the reader).
The fix in Xen's case is, on the `SYSRET` return path, to check to make sure that RCX is canonical. If it is not, Xen will fall back to the slower `IRET` path, which will behave as expected.

THE FALL-OUT

I've used "hypervisor" and "guest" because that's how it affects Xen; but it should be clear that the same exact attack might work against operating systems as well. And it turns out that it does. It seems that 64-bit versions of NetBSD, FreeBSD, and Microsoft Windows 7 were all vulnerable.

OpenBSD and Linux were not vulnerable. Linux actually fixed the bug in 2006, with CVE-2006-0744. But the description says "Linux kernel before 2.6.16.5 does not properly handle uncanonical return addresses on Intel EM64T CPUs…", which makes it sound like something Linux-specific. It's therefore not surprising that it attracted little notice from other operating systems.

Because Intel's implementation of SYSRET matches their published spec, they consider their processors to be behaving correctly. However, the `SYSRET` instruction was defined by AMD as part of the x86-64 architecture, and Intel's version is obviously intended to be compatible with AMD's version. If the majority of operating systems (acting independently) managed to "not properly handle uncanonical return addresses on Intel EM64T CPUs", it's hard not to conclude that Intel made a mistake when designing their specification.

(9)

Press Releases
(30)

Releases
(109)

Security
(10)

Technical
(12)

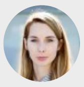
Uncategorized
(325)

User Story
(31)

RELATED POSTS

Uncategorized

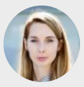
XEN PROJECT HYPERVISOR POWER MANAGEMENT: SUSPEND-TO-RAM ON ARM ARCHITECTURES



zkeaton
July 19, 2018

Uncategorized

IMPROVING THE STEALTHINESS OF VIRTUAL MACHINE INTROSPECTION ON XEN



zkeaton
June 21, 2018

Uncategorized

ANNOUNCING THE XEN PROJECT 4.11 RC AND TEST DAY SCHEDULES

Lars Kurth
April 18, 2018