EntryBleed: A Universal KASLR Bypass against KPTI on Linux

William Liu
MIT CSAIL
Cambridge, MA, USA
wliu1@mit.edu

Joseph Ravichandran
MIT CSAIL
Cambridge, MA, USA
jravi@mit.edu

Mengjia Yan
MIT CSAIL
Cambridge, MA, USA
mengjiay@mit.edu

ABSTRACT

For years, attackers have compromised systems by developing exploits that rely on known locations of kernel code and data segments. KASLR (Kernel Address Space Layout Randomization) is a key mitigation in modern operating systems which hampers these attacks through runtime randomization of the kernel image base address. KPTI (Kernel Page Table Isolation) is another defense mechanism, originally introduced to defend against the 2018 Meltdown attack by unmapping kernel addresses during user code execution. This security mechanism makes it harder for attackers to leak kernel address mappings through micro-architectural side channels. However, a few pages for system call and interrupt handling were exempted from isolation for the sake of user to kernel context transitions.

We present the EntryBleed vulnerability (CVE-2022-4543) as a universal bypass against the KASLR protection mechanism through a combination of micro-architectural side channels and design flaws in the KPTI mitigation on Intel CPUs. We demonstrate that the bug we identified can accurately de-randomize the kernel address space within a second on modern Intel CPUs in both physical host and hardware-accelerated virtual machine environments. We then provide a root cause analysis to locate the core micro-architectural behaviors that enable EntryBleed, both on physical and under virtualized environments. Furthermore, we propose a performant mitigation based closely upon a pre-existing KASLR hardening mechanism. If left unpatched, attackers will be able to easily bypass KASLR, greatly lowering the barrier for exploit development and increasing the risk of serious threats against the Linux operating system.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**; **Operating systems security**.

KEYWORDS

micro-architecture, side-channel, Linux kernel, ASLR, KPTI

ACM Reference Format:
William Liu, Joseph Ravichandran, and Mengjia Yan. 2023. EntryBleed: A Universal KASLR Bypass against KPTI on Linux. In *Hardware and Architectural Support for Security and Privacy 2023 (HASP '23), October 29, 2023, Toronto, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3623652.3623669

HASP '23, October 29, 2023, Toronto, Canada

ACM ISBN 979-8-4007-1623-2/23/10.
https://doi.org/10.1145/3623652.3623669

1 INTRODUCTION

Traditionally, low-level security research has focused on memory and thread safety [29]. Researchers studied attacks and defenses against memory bugs like buffer overflows or use-after-frees and concurrency bugs like race conditions. An attacker's goal is to corrupt program memory and hijack its execution flow to gain more privileges in a victim's device.

With the discovery of the Meltdown [22] and Spectre [21] attacks in 2018, micro-architectural vulnerabilities came to the spotlight. These bugs do not result from programming errors, but rather from hardware design choices emphasizing aggressive performance optimizations. They mostly lead to side channels, i.e., attacks which leak sensitive system secrets through measurements of external and unintended side-effects of program behavior. For example, speculative execution allows instructions to execute transiently ahead of time in the CPU pipeline even if these instructions are illegal from a permission (Meltdown) or control flow (Spectre) perspective. While the hardware has a built-in rollback mechanism, it fails to rewind all modified micro-architectural states, leaving unintentional side effects that serve as sources of information leakage. Leaked secrets include cryptographic private keys [27], secure enclave contents [30], and KASLR layout [22].

Note that, among the secrets that can be leaked via micro-architectural attacks, KASLR layout is of special importance due to its role in kernel hardening, as it is a resilience measure against software exploits. In fact, Meltdown [22] can be used to leak sensitive program data across security boundaries, including kernel pointers that could lead to a KASLR bypass. This paper's EntryBleed attack reveals TLB-resident kernel virtual addresses which can derandomize KASLR layout under the protection of KPTI, an important memory isolation mechanism between kernel and userspace as well as a KASLR hardening feature in the post-Meltdown era.

1.1 KASLR Security

KASLR [11] is an important security feature that randomizes the layout of kernel memory regions (including code, the heap, the kernel stack, etc.) on each reboot. The Meltdown attack led to major security complications as it trivially broke this randomization barrier for attackers seeking to exploit memory corruption vulnerabilities. In particular, Meltdown relies on the shared page tables (further discussed in Section 2.1) between kernel and userspace. With this sharing, the CPU can continue its aggressive speculative execution across privilege boundaries as long as any requested address is resolvable by the MMU (Memory Management Unit).

The urgent need to address this vulnerability led to the current state-of-the-art mitigation known as KPTI (Kernel Page Table Isolation). This security mechanism separates kernel and user page table entries. As a result, upon switching between privilege levels, the OS is required to switch its top-level page table pointer. Most commercial operating systems (such as Windows) have adopted similar approaches [19] in an attempt to stop micro-architectural attacks like Meltdown. Since most production operating systems utilize KPTI as a hardening mechanism, a potential failure in KPTI to protect against KASLR bypasses has serious security implications for standard user and organizational threat models.

Unfortunately, many have made overarching assumptions about the security of KPTI, overlooking that certain classes of micro-architectural attacks are still applicable as a bypass against KASLR in spite of KPTI. Even as of December 2022, the prominent security research group Google ProjectZero [18] wrote that KPTI mitigates prefetch attacks [14] across privilege boundaries. We show that this presumption is incorrect with the EntryBleed attack.

1.2 EntryBleed

We recognized a design flaw in Linux KPTI related to insufficient isolation between the userspace and kernelspace addresses. Specifically, as the OS needs to handle exceptions, interrupts, and syscalls from userspace, there are still tiny stubs of kernel addresses mapped into user page tables, serving as the entry and exit portal for userland code. We refer to this code as a **trampoline region**. We hypothesized that the trampoline region could be a leakage source for a micro-architectural-based KASLR bypass. Based on this hypothesis, we extended the prefetch-based side channel attack to construct a universal KASLR bypass. Our attack is resilient to normal operating system noise, does not require per-system configuration tuning, and works within a second.

An especially interesting question is the true root cause of this vulnerability on both physical hosts and hardware-accelerated VM environments. We found that in both contexts, the attack works because immediately before returning back from kernelspace to userland code, the TLB caches the page translation for the trampoline addresses. As such, the TLB states can then be inspected later in userspace via a prefetch side channel to leak the KASLR layout.

Furthermore, it is of interest to understand how EntryBleed works in a hardware-accelerated VM environment, especially how the micro-architectural side effects survive across guest-host context switches. We experimentally found this to be due to modern ISA optimizations on virtualized MMUs. For the purposes of this project, the scope for the VM-related analysis focuses on Intel VT-x extensions utilized in the KVM hypervisor environment, with special emphasis on the MMU optimizations EPT, VPID [17], and shadow paging, as these features are extremely common in personal and commercial computing environments. Additionally, since this vulnerability is still unpatched and exploitable in the wild, we propose an effective and performant mitigation based on pre-existing work on KASLR hardening.

Our Contributions. As of now, we have made the following contributions.

- We discovered Entrybleed, a security issue which affects current production kernels.
- We present a study of the root causes of EntryBleed in both bare metal and hardware accelerated virtualized environments.
- We provide a potential fix to this systematic vulnerability that aims to also be performant, based closely upon the pre-existing FG-KASLR [4] mitigation.

Disclosure. RedHat has publicly acknowledged EntryBleed's threat to KPTI (designated under CVE-2022-4543) [16], and other members of the security community have managed to cross-verify its success, as seen in the KASLD repository [5] for documenting KASLR bypasses.

Outline. Section 2 provides important pre-requisite knowledge for the EntryBleed attack. Section 3 describes the methodology to which we discovered the systematic flaw in KPTI and how we designed our attack along with a root cause analysis. Section 4 follows with our experimental verification of EntryBleed and Section 5 then provides data on our POC's performance, accuracy, runtime, and behavior across a variety of Intel CPUs as well as Linux kernels. We provide metrics for its behavior under different hardware-accelerated VM configurations to further our root cause analysis. Section 6 summarizes the worrying implications of this attack and suggests avenues of potential future research. Section 7 discusses our variation of the pre-existing FG-KASLR mitigation as a defensive measure, and Section 8 concludes our work.

2 BACKGROUND

2.1 Virtual Memory and Paging

To support process isolation, program portability, and memory optimizations, most CPUs support concepts known as virtual memory and paging. Rather than allowing programs to work directly with physical memory provided by DRAM, operating systems abstract it away with virtual addresses mapped to real addresses by the underlying hardware MMU. As this map is also stored in main memory, a direct translation structure would be extremely inefficient.

The concept of multi-level paging resolves this aforementioned inefficiency. On `x86_64`, virtual addresses are split into 4 or 5 sections, in which bits from each section act as an index into an array known as a page table. The address for the first section's page table is in the CR3 register [17]. Each index stores the physical address of the array for the next level's page table given the current level's index (along with metadata related to memory properties). The final section's page table stores physical addresses representing either a 4KB, 2MB, or 1GB region of contiguous memory for the virtual address. This multi-level address translation design allows for many optimizations, and even saves memory as page table entries (along with its associated memory) can be populated on demand.

Translation Lookaside Buffer. The above design works well for memory efficiency and abstractions but has one major flaw: each virtual memory access requires 3 to 5 lookups in DRAM (depending on the size of the target page), each of which takes hundreds of cycles. To address this issue, each CPU core maintains a structure known as the TLB, or Translation Look-Aside Buffer, where virtual addresses are mapped to their direct physical addresses. This structure fills up based on successful MMU resolutions of virtual addresses and can automatically evict or evict based on execution

of specific `x86_64` instructions, which the OS utilizes to maintain memory coherency.

2.2 Address Randomization

ASLR (Address Space Layout Randomization) is a common userland and kernel security feature enabled on all modern operating systems, in which the memory layout of programs is scrambled per run or boot. Before this mitigation, attackers could just hardcode virtual addresses to desirable memory targets in their exploits. The added factor of randomization often requires attackers to achieve a leakage primitive. Exploits that work independently of ASLR, or without much knowledge of the virtual address space, are also a possibility, but much less common. Aside from greatly increasing attack complexity, it can even neutralize exploitability depending on the bug.

However, in practice, the randomization only happens at the granularity of program regions such as the heap, stack, or binary image. Operating systems also impose a limit on the randomization granularity by ensuring that program regions remain within certain address ranges. For example, the Linux kernel's code and data are mapped at a 2MB boundary; combined with its allowed virtual address range, the total randomization entropy is only 9 bits [20] [6]. While the entropy is somewhat low, an exploit that cannot bypass KASLR would only work in $\frac{1}{512}$ attempts (assuming a reliance on only kernel data and code like in ROP chain attacks), which is much less dangerous and more easily detectable than a fully stable exploit. Hence, some security researchers have been interested in bypassing KASLR through micro-architectural side-channels, such as using Meltdown [22] and the double page fault attack [15].

2.3 Timing and Prefetch Side-Channels

Timing side-channels are vectors which attackers exploit to deduce secrets based on operating compute time that are data or input dependent. A very common toy example is the implementation of a naive `memcmp` for a password checker: as soon as one character is wrong, the function returns false. This password checker takes longer to finish if an attempt is more correct, thereby making runtime into an oracle for attackers, allowing them to derive the password byte by byte.

Prior work has shown that various micro-architectural structures can be used to construct side channel attacks. For example, CPUs maintain a hierarchy of memory caches to cache contents in groups of 64 or 128 contiguous bytes known as "cache lines." The CPU cache has long been used as a vector for side channels, as seen in the targeting of the L1 as well as L2 cache [27], the LLC cache [23], and full cache hierarchy in the FLUSH+RELOAD attack [31]. This is mainly because clear timing differences arise when measuring data accesses that are active in different levels of the cache hierarchy instead of just DRAM. Though less popular than caches, another micro-architectural structure for side-channels is the TLB. It has been used for side channels in the context of targeting SGX under hyperthreading [30], with the help of machine learning to deduce a victim's memory access patterns [12], or, in our case, in tandem with prefetching.

Prefetch Attack. For performance programming reasons, the ISA allows users to preemptively cache virtual memory with a family of instructions known as prefetch instructions. These instructions come with a known measurable micro-architectural side effect detectable within the granularity of CPU cycles. A given prefetch instruction will take longer depending on whether the prefetched address is mapped or not. Additionally, trying to prefetch an invalid or kernel memory address will never cause any architecturally visible exceptions, so there is no penalty for trying to prefetch a kernel address from user context. This technique is called the prefetch attack and a known bypass against standard ASLR [14].

3 THE ENTRYBLEED ATTACK

We now discuss the methodology for discovering the "EntryBleed" bug, in which we study the design of the micro-architectural defense KPTI [7] along with a source analysis of its implementation. We then provide our systematic approach to verifying the vulnerability, followed by a root cause analysis.

3.1 A Security Vulnerability in KPTI

KPTI is a defense technique that is introduced to mitigate Meltdown [22] and was believed to be effective towards prefetch attacks [18]. It works by isolating kernel and user page tables. Previously, most systems used a shared kernel and userspace page-table scheme, where the two page tables were separated by only a permission bit in the same page table structure. However out-of-order execution in some CPUs simply ignored the permission bit during transient execution in the pipeline [22], allowing attackers to measure micro-architectural side-effects related to higher-privileged code and data. By isolating their page tables completely with KPTI (as shown in Figure 1a), many micro-architectural attacks which leak secrets from the kernel would no longer work as the CPU is unable to pre-emptively process untranslatable addresses.

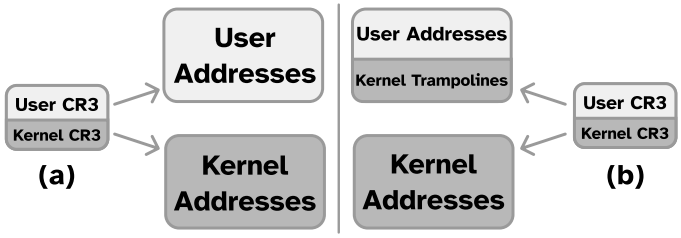


Figure 1: (a) is the ideal representation of KPTI, in which userland and kernel page tables are completely separated. (b) is the reality, as the userland needs to have kernel trampolines for proper OS functionality.

However, we found there are noticeable points of isolation failure in both its design documentation and its implementation in Linux. When executing in userspace, a minimal subset of kernel addresses is still mapped for the sake of trampolining execution into the kernel when handling interrupts, exceptions, and syscalls. This region mostly serves as a way for userland to enter and exit kernelmode, and its mapping is available in the userspace (as shown in Figure 1b). Indeed, in the Linux kernel source, the address of the syscall handler `entry_SYSCALL_64` is mapped into the `LSTAR` register in the function `syscall_init()` and is available in this

trampoline region between kernel and userspace [2]. The `LSTAR` serves as the MSR (Model-Specific Register) which informs the CPU of the instruction pointer to jump to upon syscall invocations [17].

Additionally, modern ASLR design only randomizes the start of different sections (such as program code, heap, stack, etc.). Because the syscall handler is part of kernel code, its randomized address will be at a constant offset to every other address in the kernel image. As a result, leaking the syscall handler's address would reveal the address of everything else in the kernel, breaking KASLR and rendering KPTI an ineffective mitigation against prefetch attacks.

It must be noted that we were not the first to make this observation of this weakness in KPTI's isolation. The EchoLoad [6] micro-architecture attack, which relied on load stalls as a side-channel vector, noticed this as well. There have also been successful Meltdown exploits against this specific region in both Windows from BlueFrost Security Labs [10] and MacOS from RET2 [9]. Indeed, the paper that introduced the basis for KPTI [13] also noted this problem. To our knowledge, we are the first to use the prefetch side channel to exploit this vulnerability on the latest Intel KPTI-enabled Linux kernel and virtualized environments.

3.2 Attack Strategy

From the above analysis, one can theorize the following attack scenario as a universal KASLR bypass from an unprivileged user (Figure 2).

- **Cache the syscall handler in the TLB by making a syscall from userspace.** Recall that a successful virtual to physical address translation will result in TLB caching. Upon returning from kernel space, the CPU still needs to execute a series of epilogue instructions to return back to userspace, forcing this handler to remain cached in the TLB. This works despite the effects of a TLB flush caused by the user to kernel page table register switch.
- **Guess the kernel address for `entry_SYSCALL_64` and prefetch it.** Any address in the shared page table region should work too. As mentioned earlier, this instruction has noticeable side effects on execution latency based on whether the address is cached in the TLB.
- **Iterate through all possible virtual addresses for the syscall handler based on the virtual address range for the kernel image, logging the execution cycles of prefetch for each.** The shortest measured latency implies that the virtual address is in the TLB, and is the address of the syscall handler page.

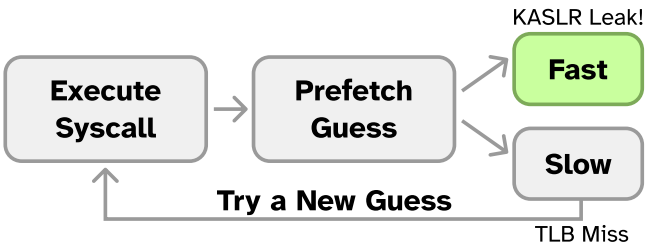


Figure 2: Visualization of the EntryBleed attack strategy.

To properly calculate CPU cycle latency, one can rely on the `rdtscp` instruction, which returns the value of a CPU clock cycle counter (the finest grained timer accessible to users of any privilege). Additionally, optimizations can be applied to the possible address space based on an analysis of Linux kernel memory mapping, and out of order execution scenarios that inaccurately skew the data can be prevented with serializing instructions like `cpuid` or `mfence`.

3.3 Root Cause Analysis

Aside from the design flaw discussed in Section 3.1, the root cause of this attack is simple. Looking at the code for `entry_SYSCALL_64` in `arch/x86/entry/entry_64.S` in the Linux source tree (version 6.0) [2], we can see that this part of kernel code starts and ends execution when the CR3 register is still holding the user's page table. Hence, the effects of the CR3 switch into kernel space address, which should flush the entire TLB, becomes nullified by the final switch back into user space CR3, thereby keeping this section of memory cached back into the TLB. This loss of state in the TLB is further avoided by the fact that this page is marked with the global bit for performance reasons, which is an `x86` feature to avoid TLB flushes on specified pages during a root page table pointer switch. Overall, this type of behavior cannot be easily patched from the entry handler's perspective due to its need to start somewhere in user space to trampoline into the kernel, nor can it easily be patched in the hardware level without some serious modification to the ISA in regards to prefetch semantics.

As mentioned in the introduction, this micro-architectural attack also functions just as well when under hardware virtualization (specifically tested on Intel VT-x). The root cause is of the same reason, but the functionality of this attack even when faced with VM exits during the side-channel procedure is an interesting observation we made that will be discussed later in Section 5.4.

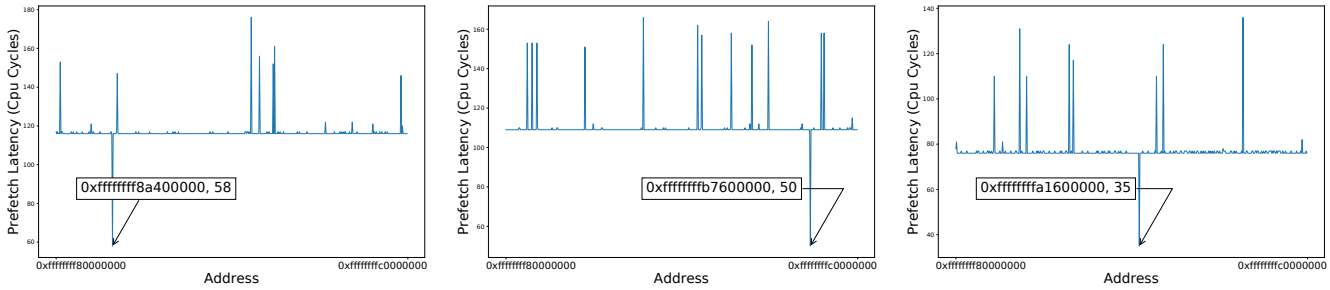
4 EXPERIMENTAL VERIFICATION

To verify the functionality of our proposed micro-architectural attack, we utilized Intel-based systems ranging from 4th generation Haswell architecture to 9th generation Coffee Lake architecture. We did not test chips 10th generation and onwards as they have hardware mitigations against Meltdown built in [8]. With these mitigations Linux automatically disables KPTI; a prefetch attack is already known to work in this case as now the kernel and user share one page table again [14] [18].

The EntryBleed exploit was tested on Linux kernel builds with hardening settings standard for both personal and cloud computing purposes, including KPTI and KASLR, the two main victims in this attack. Several of systems were also popular Linux distributions, equipped with built-in micro-architectural hardening measures such as retpolines and the latest Intel microcode updates. We also relied on Linux's KVM hypervisor driver to test the attack under hardware accelerated virtualization environments.

To accurately replicate the scenario of an attacker attempting to perform LPE (Local Privilege Escalation), we created a standard low-privileged user account and ran code that performed the attack described previously in Section 3.2.

We then transferred the attack binary over and ran it to leak the address of `entry_SYSCALL_64`, and confirmed the results through



(a) 5.4.0-146-generic + Intel i5-4590 **(b) 5.15.0-67-generic + Intel i7-6700** **(c) 5.15.0-83-generic + Intel i7-9750H**

Figure 3: Visualization of prefetch CPU cycles when side-channeling for the address of entry_SYSCALL_64

Table 1: Successful Experimental System Configurations
*** Cloud service (Digital Ocean) did not provide information for exact CPU model**

CPU Model	Kernel Build	System Environment	Tested under KVM
Intel i5-8265U	Arch 6.0.12-hardened1-1-hardened	PC	Yes
Intel i7-9750H	5.15.0-83-generic/custom 5.18.3	PC	Yes
Intel i7-9700F	6.0.12-1-MANJARO	PC	Yes
Intel i7-6700	5.15.0-56-generic	Server	No
Intel i5-4590	5.4.0-146-generic	Server	No
Intel Xeon CPU E5-2640	5.10.0-19-amd64	Cloud	No
(DO) Intel Xeon Skylake*	5.4.0-139-generic	Cloud	No

the `/proc/kallsyms` pseudo-file interface as a higher privileged user. As discussed in Section 3.1, the offset of `entry_SYSCALL_64` is at a constant offset relative to the kernel base symbol `startup_64` so its leakage effectively breaks KASLR for any given kernel build.

In the end, we developed a POC that was around 100 lines of C, and it successfully leaked KASLR base under a second with a high degree of accuracy. One can find our original reference implementation and security report at https://www.openwall.com/lists/oss-security/2022/12/16/3 [25] and https://www.willsroot.io/2022/12/entrybleed.html [26], or an updated version that achieves higher accuracy in Appendix A.

5 RESULTS

We now present an analysis of the effectiveness of EntryBleed across a variety of system configurations, demonstrating its near perfect accuracy and quick performance across many systems. We also present a root cause analysis of the EntryBleed side channel mechanism on bare metal and insights on how hardware virtualization optimizations affect the attack.

5.1 Observable Effects of EntryBleed

Figure 3 showcases clearly observable effects of the EntryBleed attack for an attacker attempting to bypass KASLR on modern Linux kernels with KPTI enabled.

In each of the graphs in Figure 3, the prefetch leakage code used ran for 1000 times at each KASLR address granularity (which should be `0x200000`) and bounded the search range from `0xffffffff80000000` to `0xffffffffc0000000`, the `x86_64` range of possible KASLR bases [3]. As shown in the graphs in Figure 3 (which relate a potential kernel virtual address to its prefetch instruction CPU execution latency), there is a noticeable drop in latency from over 100 cycles to around 35 to 60 cycles at each measurement of the `entry_SYSCALL_64` region. The observed latency drop is due to the mapped address being cached in the TLB, preventing the need for a page table walk. The address for `entry_SYSCALL_64` experienced the first major drop in latency on all the systems we analyzed.

5.2 Scope of Vulnerability

We tested the POC on the following Intel CPU models across the Linux kernel versions as seen in Table 1. Note that AMD CPU models were not considered in our attack as they were never vulnerable to Meltdown, so KPTI would have never been enabled under normal conditions for those systems.

5.3 Accuracy and Performance of EntryBleed

In Table 2 we demonstrate the attack's accuracy and speed. This is in stark juxtaposition to many other attacks in the literature which require much longer times (ranging from minutes to hours) in laboratory conditions, oftentimes requiring a post data analysis period to extract meaningful results. When run under normal system conditions, EntryBleed finishes in under half a second with effectively 100% accuracy by only taking simple averages of 1000 latency measurements, and works just as well under standard virtualization configurations. Even if the accuracy rates were to decrease on a noisier system, an attacker can easily re-run the attack for as many times as needed to confidently deduce KASLR base due to its speed.

Table 2: Average time to leak KASLR and accuracy rate of EntryBleed (per 1000 runs of POC)

CPU Model	Kernel Version	Average Leakage Time (s)	Accuracy Rate
Intel i5-4590	5.4.0-146	0.2236	100%
Intel i7-9750H	5.15.0-83	0.2761	99.7%
Intel i7-6700	5.15.0-67	0.1334	99.6%
Intel i7-9750H (KVM)	5.15.0-58	0.4148	99.9%

Another interesting question was in regards to the number of iterations needed for an accurate leakage (the default number of iterations during testing so far was 1000 as seen in Appendix A). We computed the accuracy of running the tests repeatedly to study this, starting from just 1 iteration of the prefetch attack up to 100. We repeated the test 50 times for each sample and tracked the number of correct KASLR leaks out of 50 attempts for the given iteration. As shown in Figure 4, EntryBleed can achieve a perfect success rate from just a single iteration, making it a remarkably efficient and effective attack.

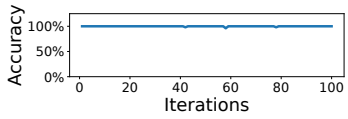


Figure 4: Relationship between prefetch iterations to the accuracy of EntryBleed

5.4 Analysis of Virtualization Behavior

Lastly, we analyzed EntryBleed in the context of Intel VT-x (Intel's virtualization technology) in Linux KVM in relation to VM relevant MMU optimizations: EPT (Extended Page Tables), VPID (Virtual Processor ID), and shadow MMU. EPT and VPID can be toggled off during load time of the KVM driver, and shadow MMU automatically activates when EPT is off. In this final case, the host maintains a "shadow" page table mapping guest virtual to host physical addresses that updates based on changes to a guest page table.

As we are interested in the preservation of the side channel's side effects across guest-host context switches, we need to force unconditional VM exits from our userland code. Otherwise, as long as there is no VM exit, it makes sense for hardware accelerated VMs to preserve the entries in the TLB as it is not switching between guests or to host. To do this, we injected a `cpuid` instruction after the syscall but before the prefetch measurement function, as `cpuid` triggers unconditional VM exits [17]. Figure 5 presents EntryBleed metrics in relation to different VM configurations.

Based on Figure 5a, we see that EPT (Extended Page Tables) does not help preserve EntryBleed's side effects across VM exits by itself. This makes sense, as EPT just acts as a second layer page table; each virtual address access not in the TLB triggers a page table walk for guest virtual to guest physical address based on the guest CR3 register, which then triggers a page table walk from guest physical to host physical address based on the EPT base pointer register in the VMCS (Virtual Machine Control Structure) [17]. EPT in general does not affect the TLB state, aside from storing a cached address translation after a successful page table walk (which shadow paging also performs).

In contrast, VPID (Virtual Processor ID) plays a major role in preserving the side effects of this side channel attack, as seen in Figure 5b and Figure 5c. VPID allows the TLB to cache address translations for multiple address spaces (similar to Intel's PCID technology or ASIDs in other architectures) [17]. The CPU can choose which TLB to use depending on its execution context and avoid TLB flushes when switching between guest and host. Even if a VM exit triggers between the syscall and prefetch measurement of EntryBleed, the guest TLB would not flush as the host TLB is in a separate VPID space.

In Figure 5d, we observe an interesting effect for when there is only shadow paging without VPID. Somehow, EntryBleed can still observe its effects on the TLB and successfully carry out a prefetch attack, albeit with a much smaller latency difference from an incorrect guess. Currently, the root cause analysis for this phenomenon is unknown to us, but we suspect there might be a multitude of cache related micro-architectural subtleties at play here as well as hypervisor software-specific optimizations. It is still an open question that requires more investigation, either through system performance counters or modification of KVM source.

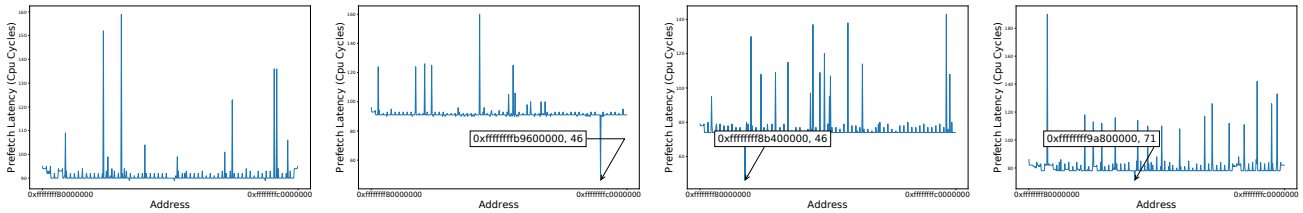
6 DISCUSSION

6.1 Implications of EntryBleed

The ability to trivially leak KASLR at a nearly perfect success rate across many systems has serious implications for the state of Linux kernel hardening. As mentioned previously, KASLR is a major barrier for many exploits targeting kernel software bugs, with attackers often going to great lengths to obtain address leaks through other corruption bugs and maintain stability of the kernel to continue exploitation. A common scenario is to reuse the same bug for both an address leak and arbitrary memory write or control flow hijacking for privilege escalation, as seen in CVE-2022-0185 [24]. EntryBleed effectively cuts the work of an attacker in half, and can revive the exploitability of bugs previously thought to be unexploitable due to KASLR, such as the "Lord of the IO_Urings" bug (CVE-2022-29968) [28]. Unlike many other documented KASLR bypasses, EntryBleed is more universal as it is independent of system misconfigurations or special user privileges – in fact, there are no software system settings or available Intel `x86` chips that can prevent this. It also finishes in under a second with nearly complete accuracy and can be re-run for as many times as needed due to its simplicity, in contrast to many other micro-architectural attacks. Devices running Linux on Intel systems face a grave and realistic risk against simpler LPE attacks.

6.2 Future Work

EntryBleed has only been thoroughly explored on Linux systems, and remains untested on Windows, Darwin, or BSD based systems. Given that most systems adopted a similar approach to KASLR and KPTI, it would be unsurprising to see similar results there, and



(a) EPT Only (b) EPT + VPID (c) Shadow MMU + VPID (d) Shadow MMU Only

Figure 5: Visualization of prefetch CPU cycles under Intel VT-x on an Intel i7 9750H CPU when side-channeling for the address of entry_SYSCALL_64 on a 5.15.0-58-generic kernel. Note how no address was found in the graph when only EPT was enabled.

would further increase EntryBleed's threat. Another interesting avenue for future exploration is to see if similar prefetch semantics are vulnerable in mobile architectures like ARM and if an attack similar in style can be launched to bypass KASLR there.

7 MITIGATION PROPOSAL

One possible solution to EntryBleed would be to relocate the addresses containing kernel exception handlers during boot time, before the exception tables and relevant MSR registers are initialized. While a prefetch attack can still leak the kernel trampolines' addresses, their addresses would be at non-constant offsets to kernel base, thereby decoupling their leakage from a KASLR bypass for the rest of the kernel protected by KPTI. This one-time randomization would also add less overhead than a per process randomization of the exception handling code.

Something similar to this idea already exists in the form of FG-KASLR, in which all kernel functions are relocated at randomized offsets at boot time as an exploit mitigation. We have only been able to test the original implementation, which did not randomize assembly based functions so EntryBleed is still functional there; according to the Linux kernel mailing lists [1], the most recent version addresses this issue but we have yet to verify it. It is also known to cause about a second of delay during boot time [4], which is unacceptable for cloud based environments with the growing trend of heavy workloads related to micro-services and on demand VMs. Overall, we believe that only these exposed handlers between user space and kernel space require randomization for an adequate and effective mitigation, but do not have plans to develop a working prototype due to the extremely heavy engineering effort better suited for core developers of the Linux kernel.

Lastly, although not completely related to EntryBleed, we would advise OS vendors against disabling KPTI in CPUs with hardware Meltdown mitigations, as previous work for the prefetch attack shows. We do not expect these to be the only and last bugs that exploit a shared page table scheme between kernel and userland.

8 CONCLUSION

EntryBleed presents an efficient, noise-resilient, and system configuration independent mechanism to bypass KASLR on modern Intel based systems when running under KPTI through the usage of `x86` prefetch instructions. Due to its effectiveness and fast rate of leakage, it can significantly lower the barrier for malicious attackers looking to design kernel exploits as it removes the need for a leakage primitive for virtual address space derandomization. We also provide an analysis on how the attack can survive across VM exits under Intel VT-x, and conclude with a mitigation proposal based on pre-existing work for FG-KASLR.

REFERENCES

[1] 2021. *Function Granular KASLR.* https://lore.kernel.org/all/20211223002209.1092165-1-alexandr.lobakin@intel.com/

[2] 2023. *Linux source code (v6.0).* https://elixir.bootlin.com/linux/v6.0/source

[3] 2023. *Virtual Memory Map.* https://www.kernel.org/doc/Documentation/x86/x86_64/mm.txt

[4] Kristen Accardi. 2020. *Function-Granular KASLR.* https://lwn.net/Articles/824307/

[5] bcoles. 2023. *KASLD.* https://github.com/bcoles/kasld

[6] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. 2020. KASLR: Break It, Fix It, Repeat. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security* (Taipei, Taiwan) *(ASIA CCS '20)*. Association for Computing Machinery, New York, NY, USA, 481–493. https://doi.org/10.1145/3320269.3384747

[7] Jonathan Corbet. 2017. *KAISER: hiding the kernel from user space.* https://lwn.net/Articles/738975/

[8] Intel Corporation. 2023. *Intel Software Security Guidance.* https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/processors-affected-consolidated-product-cpu-model.html

[9] Jack Dates. 2022. *The LDT, a Perfect Home for All Your Kernel Payloads.* https://blog.ret2.io/2022/08/17/macos-dblmap-kernel-exploitation/

[10] Nico Economou. 2020. *Meltdown Reloaded: Breaking Windows KASLR by Leaking KVA Shadow Mappings.* https://labs.bluefrostsecurity.de/blog/2020/06/30/meltdown-reloaded-breaking-windows-kaslr/

[11] Jake Edge. 2013. *Kernel address space layout randomization.* https://lwn.net/Articles/569635/

[12] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 955–972. https://www.usenix.org/conference/usenixsecurity18/presentation/gras

[13] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. KASLR is Dead: Long Live KASLR. 161–176. https://doi.org/10.1007/978-3-319-62105-0_11

[14] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. 2016. Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 368–379. https://doi.org/10.1145/2976749.2978356

[15] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical Timing Side Channel Attacks against Kernel Space ASLR. In *2013 IEEE Symposium on Security and Privacy*. 191–205. https://doi.org/10.1109/SP.2013.23

[16] RedHat Inc. 2022. *CVE-2022-4543.* https://access.redhat.com/security/cve/cve-2022-4543

[17] Intel. 2020. *Intel 64 and IA-32 Architectures Software Developer's Manual: System Programming, Volume 3.*

[18] Seth Jenkins. 2022. *Exploiting CVE-2022-42703 - Bringing back the stack attack.* https://googleprojectzero.blogspot.com/2022/12/exploiting-CVE-2022-42703-bringing-back-the-stack-attack.html

[19] Ken Johnson. 2018. *KVA Shadow: Mitigating Meltdown on Windows.* https://msrc.microsoft.com/blog/2018/03/kva-shadow-mitigating-meltdown-on-windows/

[20] Taehun Kim, Taehyun Kim, and Youngjoo Shin. 2021. Breaking KASLR Using Memory Deduplication in Virtualized Environments. *Electronics* 10, 17 (2021). https://www.mdpi.com/2079-9292/10/17/2174

[21] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. 1–19. https://doi.org/10.1109/SP.2019.00002

[22] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proceedings of the 27th USENIX Conference on Security Symposium* (Baltimore, MD, USA) *(SEC'18)*. USENIX Association, USA, 973–990.

[23] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*. 605–622. https://doi.org/10.1109/SP.2015.43

[24] William Liu. 2022. *CVE-2022-0185 - Winning a $31337 Bounty after Pwning Ubuntu and Escaping Google's KCTF Containers.* https://www.willsroot.io/2022/01/cve-2022-0185.html

[25] William Liu. 2022. *CVE-2022-4543: KASLR Leakage Achievable even with KPTI through Prefetch Side-Channel.* https://www.openwall.com/lists/oss-security/2022/12/16/3

[26] William Liu. 2022. *EntryBleed: Breaking KASLR under KPTI with Prefetch (CVE-2022-4543).* https://www.willsroot.io/2022/12/entrybleed.html

[27] Colin Percival. 2009. Cache missing for fun and profit. (08 2009).

[28] Joseph Ravichandran and Michael Wang. 2022. *Lord of the io_urings.* Technical Report.

[29] László Szekeres, Mathias Payer, Tao Wei, and Dawn Song. 2013. SoK: Eternal War in Memory. In *2013 IEEE Symposium on Security and Privacy*. 48–62. https://doi.org/10.1109/SP.2013.13

[30] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 2421–2434. https://doi.org/10.1145/3133956.3134038

[31] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 719–732.

A ENTRYBLEED POC

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

#define KERNEL_LOWER_BOUND 0xffffffff80000000ull
#define KERNEL_UPPER_BOUND 0xffffffffc0000000ull
#define entry_SYSCALL_64_offset 0xe00000ull

uint64_t sidechannel(uint64_t addr) {
  uint64_t a, b, c, d;
  asm volatile (".intel_syntax noprefix;"
    "mfence;"
    "rdtscp;"
    "mov %0, rax;"
    "mov %1, rdx;"
    "xor rax, rax;"
    "lfence;"
    "prefetchnta qword ptr [%4];"
    "prefetcht2 qword ptr [%4];"
    "xor rax, rax;"
    "lfence;"
    "rdtscp;"
    "mov %2, rax;"
    "mov %3, rdx;"
    "mfence;"
    ".att_syntax;"
    : "=r" (a), "=r" (b), "=r" (c), "=r" (d)
    : "r" (addr)
    : "rax", "rbx", "rcx", "rdx");
  a = (b << 32) | a;
  c = (d << 32) | c;
  return c - a;
}

#define STEP 0x200000ull
#define SCAN_START KERNEL_LOWER_BOUND + entry_SYSCALL_64_offset
#define SCAN_END KERNEL_UPPER_BOUND + entry_SYSCALL_64_offset

#define DUMMY_ITERATIONS 5
#define ITERATIONS 1000
#define ARR_SIZE (SCAN_END - SCAN_START) / STEP

uint64_t leak_syscall_entry(void)
{
    uint64_t data[ARR_SIZE] = {0};
    uint64_t min = ~0, addr = ~0;

    for (int i = 0; i < ITERATIONS + DUMMY_ITERATIONS; i++)
    {
        for (uint64_t idx = 0; idx < ARR_SIZE; idx++)
        {
            uint64_t test = SCAN_START + idx * STEP;
            syscall(104);
            uint64_t time = sidechannel(test);
            if (i >= DUMMY_ITERATIONS)
                data[idx] += time;
        }
    }

    for (int i = 0; i < ARR_SIZE; i++)
    {
        data[i] /= ITERATIONS;
        if (data[i] < min)
        {
            min = data[i];
            addr = SCAN_START + i * STEP;
        }
        printf("%llx %ld\n", (SCAN_START + i * STEP), data[i]);
    }

    return addr;
}

int main()
{
    printf ("KASLR base %llx\n", leak_syscall_entry() - entry_SYSCALL_64_offset);
}
```