Exploring the Meltdown Vulnerability in Operating System Security CS7339 Project Report

Name: Bingying Liang

November 17 2023

Contents

1 Abstract

This detailed examination scrutinizes the Meltdown vulnerability, a critical flaw in operating system (OS) security that undermines the fundamental isolation between user applications and the kernel. By exploiting micro-architectural features of CPUs, namely speculative execution, Meltdown breaches this isolation, posing a significant threat to system integrity. Our study delves into the intricate mechanisms behind the vulnerability, evaluates the effectiveness of various mitigation strategies, and discusses the broader implications for future OS security frameworks. The objective is to provide a comprehensive understanding of Meltdown, facilitating informed developments in computing security.

2 Introduction

The integrity and security of modern computing systems are fundamental to the digital infrastructure on which society increasingly depends. The core of these systems is the operating system (OS), which manages hardware resources and provides public services to computer programs. A major security feature of today's operating systems is memory isolation -a mechanism that prevents user programs from accessing each other or kernel memory. This isolation is not only a theoretical construct but also a practical need, ensuring that multiple applications can run simultaneously on personal devices and execute the processes of multiple users on a single machine, especially in cloud environments.

However, the discovery of Meltdown vulnerability challenges the robustness of this security feature. Meltdown is a hardware vulnerability affecting modern processors that exploits the side effect of out-of-order execution to bypass memory isolation and read arbitrary kernel memory locations, including personal data and passwords. This vulnerability is independent of the operating system, and it is worth noting that it does not depend on any software vulnerability. It breaks the basic security guarantee provided by address space isolation and affects millions of devices and users worldwide.

2.1 Background of Operating System Security

For operating system security, an application is controlled that only given resources can be accessed, such as files, processes, I/O, IPC, and that given operations, such as executable or read-only, can be performed [1]. However, limited containment supports most commercial operating systems (MS Windows, various Unixes, etc.) to make access decisions based only on user identity and ownership, without considering additional security-related criteria such as operation and trustworthiness of programs, role of users, and sensitivity or integrity of data. As long as users or applications have complete discretion over objects, it is not possible to control the data flow or enforce system-wide security policies. Due to this weakness of current operating systems, once an application is compromised, for example, through a buffer overflow attack, it is easy to break the security of the entire system. Most modern information computer systems provide concurrent execution of multiple applications on a single physical computing hardware, which may contain multiple processing units. In such a multitasking, time-sharing environment, individual application jobs share the same system resources such as CPU, memory, disk, and I/O devices under the control of the operating system [1]. To protect the execution of application jobs from possible interference and attacks from other jobs, most operating systems implement some abstract properties including process and task control blocks (TCB), virtual memory Spaces, files, ports, interprocess communication (IPC), etc.

2.1.1 Principles of OS Security

As part of the overall goal of providing secure computer systems, the design of verifiably secure operating systems is one of the most important tasks. Four key [2] operating system partitions are identified: user interface functions, services invoked by the user, background services, and the security kernel. Among them, interface functions provide a safe initial environment for executing user programs, restrict users to call services, background services can not access user information, and security kernel to fully protect the stored information are also very important.

2.2 Kernel/User Space Isolation

To protect memory and hardware from malicious or faulty software behavior, operating systems for modern computers typically divide virtual memory into user space and kernel space [4], with the kernel space strictly reserved for running privileged operating system kernels, kernel extensions, and most device drivers. The user space is the memory area where the application software and some drivers execute.

2.2.1 Importance in OS Design

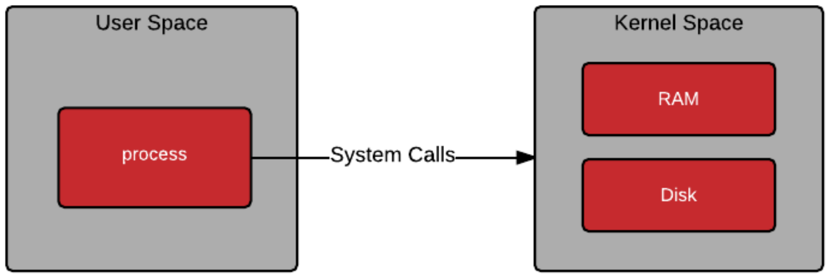

Figure 1: User Space vs Kernel Space [6]

Kernel/userspace isolation, as shown in Figure 1, is one of the key ideas in operating system design and can play a crucial role in ensuring system security, stability, and performance. This isolation can not only protect the system from various security threats, but also support the efficient and safe operation of modern multi-user and multi-task environments. And as operating systems continue to evolve, maintaining and enforcing this isolation remains a key priority in response to new challenges and technological advances.

2.2.2 Traditional Mechanisms for Isolation

For the traditional operating system kernel/user space isolation mechanism has evolved to create a secure and stable computing environment. While these mechanisms utilize hardware capabilities and software policies to enforce a clear separation between the kernel and user applications, protecting the system against various potential security vulnerabilities and system failures. For example, Denali isolation kernel, an x86-based operating system that isolates untrusted software services in separate protection domains. Architecturally, Denali is a thin software layer that runs directly on x86 hardware. Denali exposes a virtual machine (VM) architecture that is based on the x86 hardware, and supports the secure multiplexing of many VMs on an underlying physical machine. [7] As technology advances, these traditional mechanisms need to continue to be refined and supplemented with new security measures to address emerging threats and vulnerabilities.

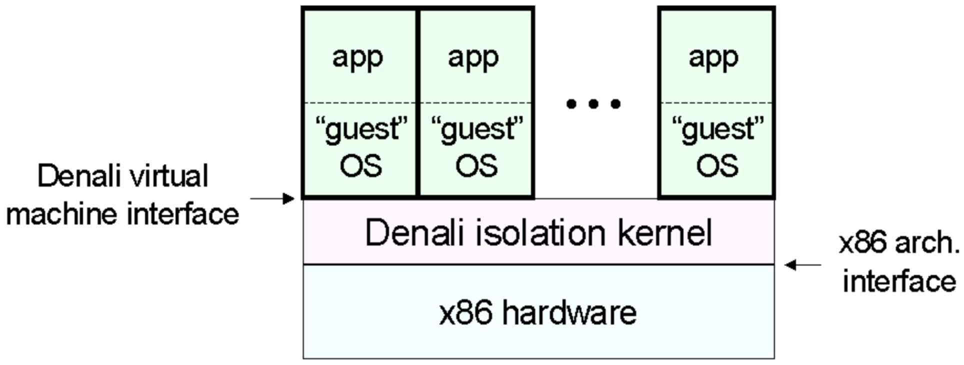

Figure 2: The Denali isolation kernel [7]

2.3 Introduction to CPU Architecture and Speculative Execution

2.3.1 Basic CPU Design Principles

Computer pioneers correctly predicted that programmers would want unlimited amounts of fast memory. An economical solution to that desire is a memory hierarchy, which takes advantage of locality and trade-offs in the cost-performance of memory technologies as figure 3 shows. [8]

For virtual memory, TLBS, first level caches, and second level caches, all mapped parts of virtual and physical address space can be confused about which bits go where. Figure 3 gives a hypothetical example from a 64-bit virtual address to a 41-bit physical address with two levels of cache. The L1 cache is virtually indexed and physically tagged since both cache size and page size are 8 KiB. The L2 cache is 4mib. Both have a block size of 64 bytes[8].

2.3.2 Role and Benefits of Speculative Execution

Maintaining control dependencies becomes an increasing burden when you want to use instruction-level parallelism. Branch prediction reduces the direct stagnation attributable to branches, but for a processor executing multiple instructions per clock, accurate branch prediction alone may not

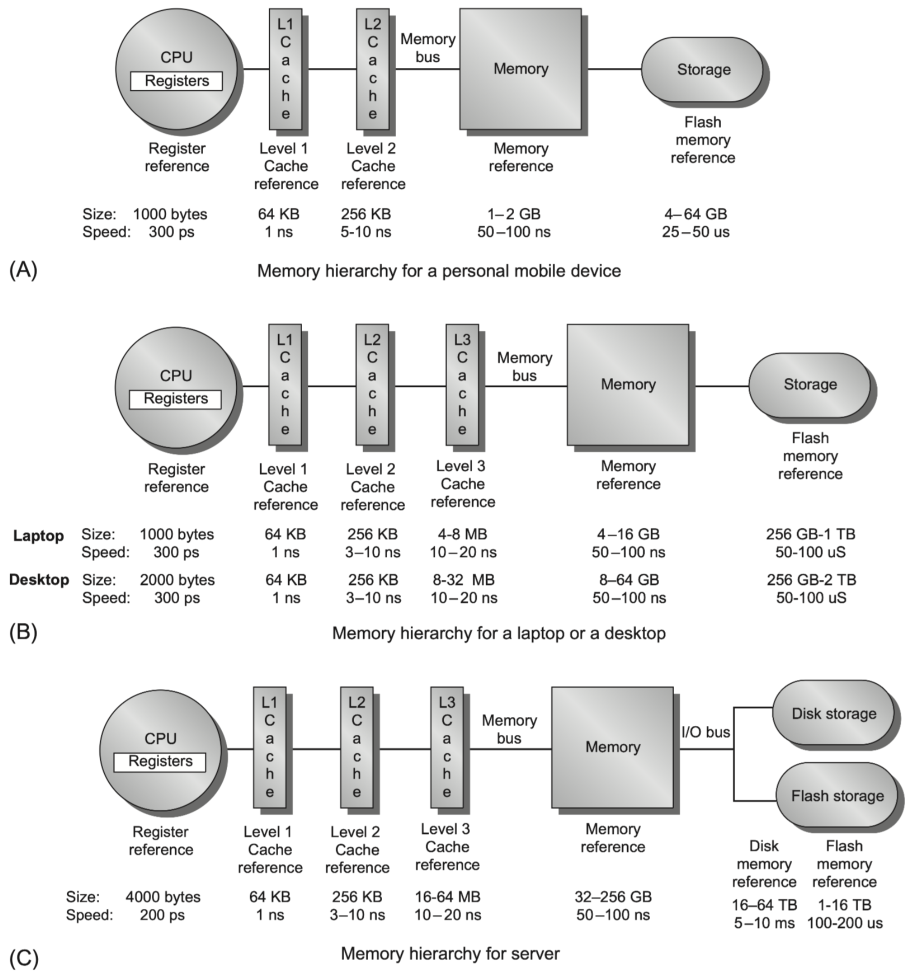
CPU
Registers
L1 Cache
L2 Cache
Memory bus
Memory
Storage
Register reference
Level 1 Cache reference
Level 2 Cache reference
Memory reference
Flash memory reference
Size: 1000 bytes · 64 KB · 256 KB · 1–2 GB · 4–64 GB
Speed: 300 ps · 1 ns · 5-10 ns · 50–100 ns · 25–50 us
(A)
Memory hierarchy for a personal mobile device
L3 Cache
Level 3 Cache reference
Laptop Size: 1000 bytes · 64 KB · 256 KB · 4-8 MB · 4–16 GB · 256 GB-1 TB
Speed: 300 ps · 1 ns · 3–10 ns · 10–20 ns · 50–100 ns · 50-100 uS
Desktop Size: 2000 bytes · 64 KB · 256 KB · 8-32 MB · 8–64 GB · 256 GB-2 TB
Speed: 300 ps · 1 ns · 3–10 ns · 10–20 ns · 50–100 ns · 50-100 uS
(B)
Memory hierarchy for a laptop or a desktop
I/O bus
Disk storage
Flash storage
Disk memory reference
Flash memory reference
Size: 4000 bytes · 64 KB · 256 KB · 16-64 MB · 32–256 GB · 16–64 TB · 1-16 TB
Speed: 200 ps · 1 ns · 3–10 ns · 10–20 ns · 50–100 ns · 5–10 ms · 100-200 us
(C)
Memory hierarchy for server

Figure 3: The levels in a typical memory hierarchy

be sufficient to produce the required amount of instruction-level parallelism. A wide-issue processor may need to execute one branch per clock cycle to maintain maximum performance. Thus, exploiting more parallelism requires overcoming the limitation of control dependence, which is accomplished by conjecturing the outcome of a branch and executing the program, a mechanism that is a subtle but important extension to branch prediction based on dynamic scheduling, and hardware speculation, which extends the idea of dynamic scheduling shows in figure5.

This paper aims to dissect Meltdown vulnerabilities deeply. Beginning with an overview of the basic concepts of memory isolation and processor design, followed by a deep dive into how Meltdown exploits these architectural elements. We also analyze Meltdown's impact on different computing environments, discuss the effectiveness of the proposed countermeasure, and explore its broader impact on operating system and hardware security. Through this analysis, this paper attempts to provide one of the most important security challenges facing modern computing and highlights the need for robust security policies in both software and hardware design.

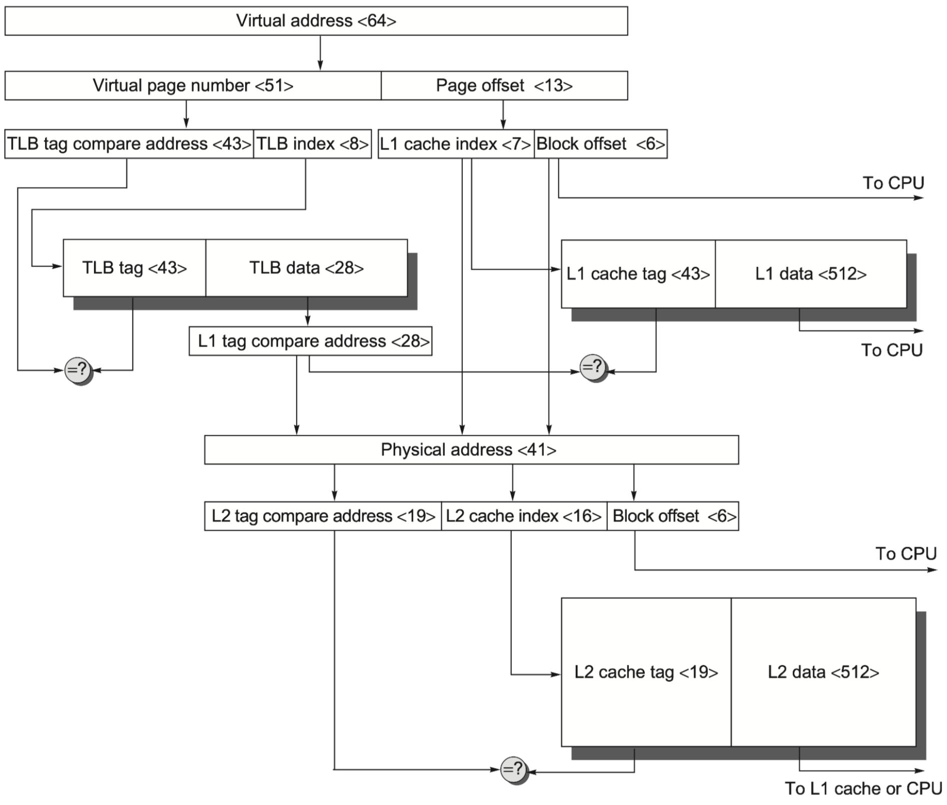

Figure 4: The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache access [8]

3 Previous Work

Prior research in operating system security has largely focused on software-based threats and defenses. The introduction of micro-architectural attacks like Meltdown represents a paradigm shift, revealing vulnerabilities within the hardware itself. We review existing literature on CPU architecture, speculative execution, and previous micro-architectural attacks to contextualize Meltdown within the broader scope of system security.

3.1 Literature Review on CPU Architecture and OS Security

Since modern processors use branch prediction and speculative execution to maximize performance, vulnerabilities can lurk. For example, if the destination of a branch depends on the memory value being read, the CPU will try to guess the destination and try to execute ahead of time. When the memory value finally arrives, the CPU will either give up or submit speculative computation, and the speed increase naturally brings some potential vulnerabilities that need to be dealt with in a timely way.

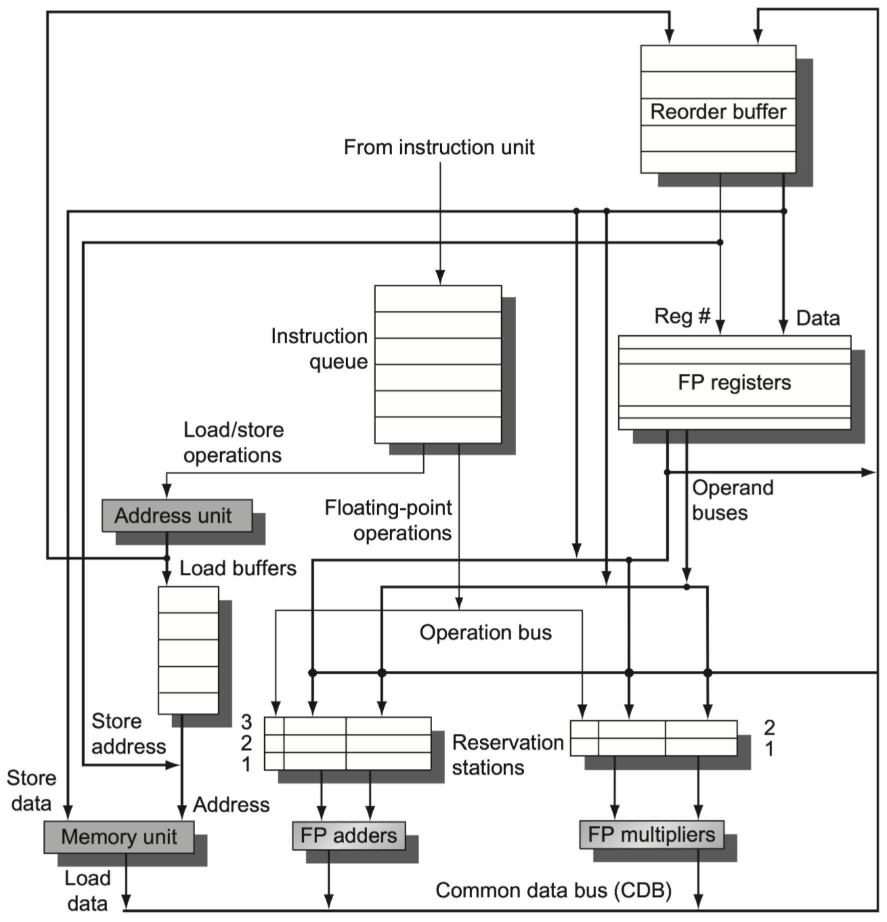

Figure 5: The basic structure of an FP unit using Tomasulo's algorithm and extended to handle speculation

3.1.1 Studies on Speculative Execution

Because the speculative logic is executed in an unfaithful way, the victim's memory and registers can be accessed, and operations with measurable side effects can be performed. And this has led to the creation of ghost attacks[5] that involve inducing a victim to speculatively perform actions that would not have occurred during correct program execution, and leaking confidential information of the victim to the adversary through side channels. This leads to side-channel attacks, fault attacks, and practical attacks on return programming oriented methods that can read arbitrary memory from the victim process. Meltdown differs from the Spectre attack in several ways, notably that Spectre requires customization to the software environment of the victim process, but applies more broadly to cpus, and KAISER cannot mitigate it[12].

3.1.2 Historical Micro-architectural Attacks

Microarchitectural side-channel attacks refer to side-channel attacks that exploit information leakage from the hardware infrastructure itself. This attack can be found in a wide range, for example, it will occur on servers, workstations, laptops, smartphones, etc[10]. Typically, we assume that the software infrastructure is secure, meaning that there are no software bugs such as buffer overflows. However, such an assumption does not imply safe execution, since information leakage is caused by the implementation, which is often driven by complex optimizations and designs. The trade-off between optimizing certain aspects of execution and potential information leakage is, for example, that kernel and user addresses will be mapped instead of switching page tables when CPU speeds are desired.

3.2 Evolution of Operating System Security

3.2.1 Shift from Software to Hardware Vulnerabilities

Due to trust in isolation, initial focus on software: Discussion Initial operating system security focused on software vulnerabilities such as buffer overflows, SQL injections, and malware. The shift to hardware concerns, where vulnerabilities such as CPU instruction set vulnerabilities can be applied to software, shifts the focus to hardware vulnerabilities, a shift triggered by the growing awareness that cpus and other hardware components can be exploited. Discuss how this shift represents a new paradigm for system security. Integrating Hardware Security in Operating System Design: Examine how this shift has affected operating system design, leading to a greater emphasis on integrating hardware security considerations in OS architectures.

3.2.2 Impact of Micro-architectural Flaws

Cpu-level vulnerabilities can undermine the most robust software security measures and thus the widespread impact of microarchitectural flaws on system security. This microarchitectural flaw poses unique challenges to mitigation measures, including the need for software patches and hardware redesign. This will greatly affect the future of operating system security. Hardware level vulnerabilities, the future trajectory of operating system security all need to be fixed. Software and hardware considerations are covered in order to mitigate such deficiencies, as well as a more comprehensive approach that may be taken to system security. This allows different microarchitectures to increase speed while keeping a close eye on vulnerabilities.

4 My Research

4.1 Detailed Analysis of the Meltdown Attack

The security of computer systems fundamentally depends on memory isolation. However, meltdown makes isolation unreliable under certain circumstances. For example, kernel address ranges are marked unreachable and prevented from being accessed by users. Meltdown[9] exploits the out-of-order side effect of modern processors to read arbitrary kernel memory locations, including personal data and passwords. Out-of-order execution is an indispensable performance feature that widely exists in modern processors. This attack is independent of the operating system and does not rely on any software vulnerabilities. Meltdown breaks all the security guarantees provided by address space isolation and paravirtualized environments. Therefore, all the security mechanisms built on this foundation are broken. On the affected system, an attacker can read the memory of other processes or virtual machines in the cloud without any permissions or privileges, thus affecting millions of customers and almost every personal computer user.

I will use an example to explain Meltdown: Meltdown happens because the kernel is mapped into the address space of each user process shows in figure 6. In the page tables, the user code runs the full kernel PTE, but they have PTE_U cleaned out, so the user code will get an error if it tries to use the kernel virtual address. All these mappings are there, they just cannot be used by user code when userspace executes[11], or cause errors when used by user code. The reason that operating system designers map both kernel and user addresses when running user code is that this makes system calls considerably faster. Intel CPUs use this approach to increase speed. Because this means that you do not have to switch the page tables when a system call occurs the page tables themselves take time. It also often causes CPU cache flushing, making subsequent code slower. The attack relies on this habit.

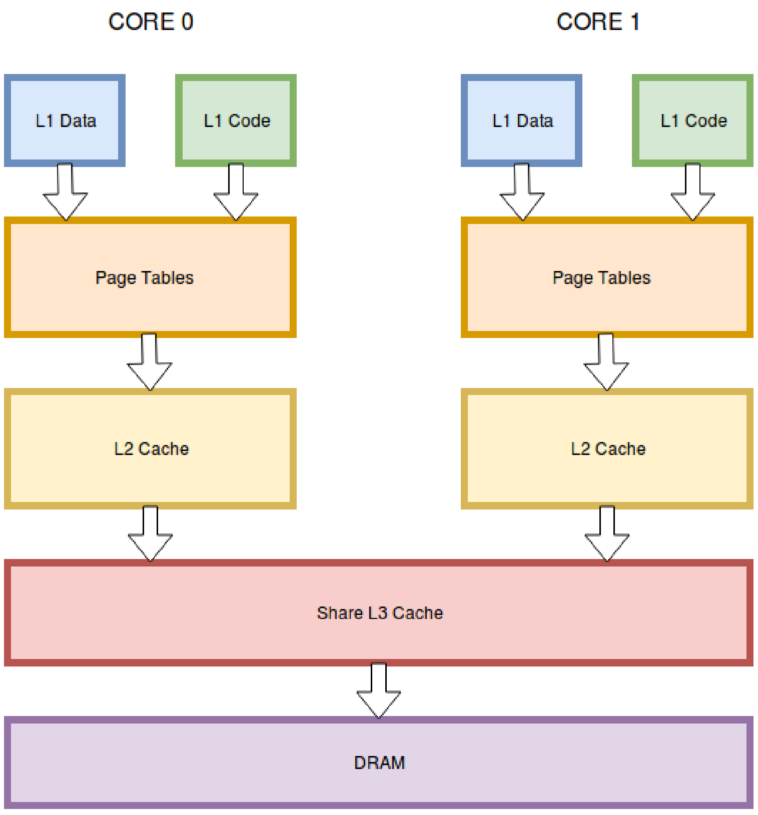

Figure 6: Simplified version of memory subsystem

```
char buf[8192]
clflush buf[0]
clflush buf[4096]

<some expensive instruction like divide>

r1 = <a kernel virtual address>
```

```
 8
 9      r2 = *r1
10      r2 = r2 & 1
11      r2 = r2 * 4096  // speculated
12      r3 = buf[r2]    // speculated
13
14      <handle the page fault from "r2 = *r1">
15
16      a = rdtsc
17      r0 = buf[0]
18      b = rdtsc
19      r1 = buf[4096]
20      c = rdtsc
21      if b-a < c-b:
22          low bit was probably a 0
```

The line 1 means only one bit is taken out of the kernel, which is multiplied by 4096, and then line 2 and line 3 means making sure the relevant part is not cached and is far away, which may be affected by hardware preloading. Line 5 is supposed use some expensive instructions that won't be completed for a long time. Line 7 will load the kernel address. Hoping to be able to predict the execution of other instructions, before this one exits, before it causes a failure and cancelates those instructions. Loading needs some time, so we can use the time to keep running the line 9, 10, 11, 12. Even though r1 is an address to which we have no authority. Since it is the kernel address, we will still see its value loaded into r3. The bad thing happens, isolation breaking.

Meltdown does not work on AMD cpus, even though the manuals for AMD cpus are the same instruction set, i.e., the same thing, as for Intel cpus. This attack does not work on AMD cpus. It is widely believed that the reason is that AMD CPU, even when predicted execution, if you do not have permission to read this address, will not even predict to load the value into r3, that is why the attack does not work on AMD CPU.

4.2 Experimental Data and Theoretical Analysis

The fact is that attacks don't always work, and Moritz Lipp's team did an experiment where they ran an attack on their own machine, the result shows in figure 7, and they extracted a bunch of data from the kernel on their own machine, and they could see that they must have retried this version of the attack a lot of times, and he mentioned performance, which means that under certain circumstances, They can extract data at a rate of only 10 bytes per second, which means they are there trying and trying again and again, and after thousands of times they finally manage to get some data, i.e. flush and reload indicates that the two cache lines have different load times. So there are some unexplained things about why meltdown fails with such a high frequency. People aren't sure what the conditions are for it to succeed or not, but you know that the most immediate possibility is that meltdown succeeds if the kernel data is in the L1 cache. This will not succeed if the kernel data is not in the L1 cache. This breaks the so-called isolation

```
79cbb80: 6c4c 48 32 5a 78 66 56 44  73 4b 57 39 34 68 6d |lLH2ZxfVDsKW94hm|
79cbb90: 3364 2f 41 4d 41 45 44 41  41 41 41 41 51 45 42 |3d/AMAEDAAAAAQEB|
79cbba0: 4141 41 41 41 41 3d 3d XX  XX XX XX XX XX XX XX |AAAAAA==........|
79cbbb0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbc0: XXXX XX 65 2d 68 65 61 64  XX XX XX XX XX XX XX |...e-head.......|
79cbbd0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbe0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbbf0: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc00: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc10: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc20: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc30: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc40: XXXX XX XX XX XX XX XX XX  XX XX XX XX XX XX XX |................|
79cbc50: XXXX XX XX 0d 0a XX 6f 72  69 67 69 6e 61 6c 2d |.......original-|
79cbc60: 7265 73 70 6f 6e 73 65 2d  68 65 61 64 65 72 73 |response-headers|
79cbc70: XX44 61 74 65 3a 20 53 61  74 2c 20 30 39 20 44 |.Date: Sat, 09 D|
79cbc80: 6563 20 32 30 31 37 20 32  32 3a 32 39 3a 32 35 |ec 2017 22:29:25|
79cbc90: 2047 4d 54 0d 0a 43 6f 6e  74 65 6e 74 2d 4c 65 | GMT..Content-Le|
79cbca0: 6e67 74 68 3a 20 31 0d 0a  43 6f 6e 74 65 6e 74 |ngth: 1..Content|
79cbcb0: 2d54 79 70 65 3a 20 74 65  78 74 2f 68 74 6d 6c |-Type: text/html|
79cbcc0: 3b20 63 68 61 72 73 65 74  3d 75 74 66 2d 38 0d |; charset=utf-8.|
```

Listing (3) Memory dump showing HTTP Headers on Ubuntu 16.10 on a Intel Core i7-6700K

Figure 7: Memory dump showing HTTP Headers on Ubuntu 16.10 on a Intel Core i7-6700K [Meltdown]

4.3 Countermeasures

KASLR[12] can be used to fix Meltdown, and something similar is currently called kpti in Linux. It's a fairly straightforward idea, the idea is not to put kernel mappings in user page tables, and to switch page tables during system calls, as in xv6 OS, which means KASLR is a kernel modification to not have the kernel mapped in the user space[13]. So only the user mapping in userspace initiates the system call, which causes the attack to not work.

4.4 Further Discussion

For predicting CPU design changes, speculative execution needs to be revisited and its prospects in future CPU design should be discussed, considering whether it will be limited, modified, or replaced with alternative methods to balance security and performance. While the integrated security features section, discusses the possibility that future cpus may incorporate more advanced built-in security features, possibly borrowing from secure enclave techniques to isolate sensitive operations more effectively.

For the evolution of operating system security models, it is necessary to adapt to hardware vulnerabilities in time and explore how operating system security models may evolve in response to the knowledge of hardware vulnerabilities. This could include more dynamic, adaptable security measures able to cope with emerging threats.

A close collaboration between hardware and software is also required, which is intended to highlight the growing need for closer collaboration between hardware manufacturers and software developers to ensure system security, possibly leading to a more integrated security approach in both areas.

For future perspectives, an in-depth exploration of how emerging technologies such as quantum computing, artificial intelligence, and the Internet of Things affect CPU architecture and operating system security is needed. Consider the new opportunities and challenges that these technologies may bring. Our ongoing focus on resilience emphasizes that cpus and operating systems are designed not just for optimal performance, but also for resilience against evolving security threats, ensuring that they can adapt and respond to unforeseen leaks

5 Conclusions

Meltdown poses a major challenge to conventional beliefs about operating system security. It exploits out-of-order execution and side channels on modern processors to read arbitrary kernel memory from programs in unprivileged userspace. Meltdown allows an attacker to read sensitive data of other processes or virtual machines in the cloud at speeds up to 503 KB/s, affecting millions of devices. The vulnerability not only exposes the limitations of existing security models, but also highlights their vulnerability, the complexity of modern CPU architectures. A wealth of information suggests that while software mitigables like KAISER/KPTI are effective, a comprehensive approach, including hardware-level fixes, is critical for long-term safety. The study highlights the ongoing arms race between system security and emerging threats, underscoring the need for constant vigilance and innovation. As technology continues to evolve, so do the challenges and threats that come with it. It is our collective responsibility and adaptation to stay ahead of the curve, stay vigilant, and foster a culture of continuous learning. While the study revealed specific vulnerabilities and defenses, its broader message is clear: complacency is not an alternative in the ever-changing cybersecurity landscape.

References

[1] Cui-Qing Yang, ”Operating System Security and Secure Operating Systems”, Accessed: Nov. 16, 2023. [Online]. Available:https://www.giac.org/paper/gsec/2776/operating-system-security-secure-operating-systems/104723.

[2] G. R. Andrews, “Partitions and principles for secure operating systems,” in Proceedings of the 1975 annual conference on ACM 75, Not Known: ACM Press, 1975, pp. 177–180. doi: 10.1145/800181.810311.

[3] T. A. Linden, “Operating System Structures to Support Security and Reliable Software,” ACM Comput. Surv., vol. 8, no. 4, pp. 409–445, Dec. 1976, doi: 10.1145/356678.356682.

[4] “User space and kernel space,” Wikipedia. Oct. 02, 2023. Accessed: Nov. 16, 2023.

[5] P. Kocher et al., “Spectre Attacks: Exploiting Speculative Execution.” arXiv, Jan. 03, 2018. Accessed: Nov. 17, 2023. [Online]. Available: http://arxiv.org/abs/1801.01203

[6] “Architecting Containers Part 1: Why Understanding User Space vs. Kernel Space Matters.” Accessed: Nov. 16, 2023. [Online]. Available: https://www.redhat.com/en/blog/architecting-containers-part-1-why-understanding-user-space-vs-kernel-space-matters

[7] “Isolation Kernel Design Principles.” Accessed: Nov. 16, 2023. [Online]. Available: https://www.usenix.org/legacy/publications/library/proceedings/osdi02/tech/full_papers /xxdiscards/whitaker_html/node5.html

[8] J. L. Hennessy, “Computer Architecture: A Quantitative Approach”.

[9] M. Lipp et al., “Meltdown: reading kernel memory from user space,” Commun. ACM, vol. 63, no. 6, pp. 46–56, May 2020, doi: 10.1145/3357033.

[10] “Introduction to micro-architectural attacks,” Coursebook for Attacks on Implementations. Accessed: Nov. 17, 2023. [Online]. Available: https://orenlab.sise.bgu.ac.il//AttacksonImplementationsCourseBook/06_Cache_Attacks _Guest_Lecture.html

[11] Jann Horn, “Project Zero: Reading privileged memory with a side-channel,” Project Zero. Accessed: Nov. 17, 2023. [Online]. Available: https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html

[12] GRUSS, D., LIPP, M., SCHWARZ, M., FELLNER, R., MAURICE, C., AND MANGARD, S. KASLR is Dead: Long Live KASLR. In International Symposium on Engineering Secure Software and Systems (2017), Springer, pp. 161–176.

[13] D. Gruss, M. Lipp, M. Schwarz, R. Fellner, C. Maurice, and S. Mangard, “KASLR is Dead: Long Live KASLR,” in Engineering Secure Software and Systems, vol. 10379, E. Bodden, M. Payer, and E. Athanasopoulos, Eds., in Lecture Notes in Computer Science, vol. 10379. , Cham: Springer International Publishing, 2017, pp. 161–176. doi: 10.1007/978-3-319-62105-0_11.