International Journal of Applied Engineering Research ISSN 0973-4562 Volume 15, Number 6 (2020) pp. 546-549

Exploitation of Meltdown on x86

Kshitij Tiwari, Mallappa Bagi, Dr. Azra Nasreen, Dr. Pratiba D, Jyoti Shetty

Department of Computer Science and Engineering, R V College of Engineering, R V Vidyanikethan Post, Bengaluru - 560059, India.

Abstract

Spectre and Meltdown were discovered in 2018 and have spawned a new family of vulnerabilities. Meltdown breaks access violation semantics by stealing the unauthorized data. This paper explores Flush+Reload and Meltdown implemented in C. Meltdown is run on 32 bit virtual machine. It is observed that the mitigations preented for Meltdown are stop-gap measures to fix its symptoms without addressing the root cause.

Keywords: Meltdown, Flush+Reload, side channel, virtual memory, memory protection, kernel modules, procfs, virtual machine, Linux, TSC, speculative execution, KPTI, microcode, Haswell

1. INTRODUCTION

Memory isolation is an important component of modern OS, and enforced by virtual memory systems. User space process can only access user space addresses in virtual memory. Attempting to access the other locations results in exception. Meltdown [1] is a security vulnerability where unprivileged process can steal data memory location mapped in virtual address space. Hence, Meltdown "melts down" the security barrier provided by virtual memory semantics, and dismantles the guarantees made by the System Software and the CPU. In this paper, Meltdown is exploited on 32 bit Linux system. The approach for implementing the attack, allocating memory, enforcing data locality and handling exceptions is defined. A user-space process is shown to leak data from the kernel memory. Results are presented followed by the analysis of the mitigations.

2. RELATED WORK

Side channel attacks utilize information produced as a by-product of the operation of physical devices to steal information. Traditional attacks involve measuring electromagnetic radiation, power consumption, or using acoustics to record data such as key strokes. With the discovery of the Flush+Reload attack [2], interest in timing analysis as a side channel attack has increased substantially. It was clearly demonstrated that given a sufficiently accurate timing mechanism, one could leak the recently used data with the processor cache serving as a side channel. The approaches in [3] and [4] outline exploitations of such attacks. Extending the idea of Flush+Reload, Meltdown and Spectre [5] are aimed at breaking down the security guarantees of the CPU by allowing unprivileged processes to steal data from privileged memory locations. This grants a user-space process access to another user-space process's memory, or access to the kernel-space memory. Access to kernel space memory is particularly dangerous as it contains the page table and the TLB, which contain the per-process memory-mappings. Both of these attacks exploit the side effects of speculative execution semantics in the processor. Speculative execution [6] is a technique used by most modern processors to enhance the performance. This involves premature execution of independent instructions by the CPU to hide latencies. For example, a load operation stalls till the required data is loaded from the memory. This latency can be hidden by saturating the ALUs with instructions that do not depend on the loaded value and keeping their results ready for commit. Critical flaws in the implementation of speculation on most Intel and some ARM CPUs gives rise to Meltdown attack. Spectre attack combines speculative execution and control flow prediction. All modern-day CPUs are vulnerable to Spectre. Several similar vulnerabilities such as Foreshadow, the MDS family, Zombieload 2, and Plundervolt, have been uncovered. All of them follow the same basic principle of spectre and meltdown, using the cache as a covert side channel to leak data.

3. PROPOSED SYSTEM

Initially values local to the process are first stolen using cache side channel attack. First, the API for time determination is chosen. The Flush+Reload attack is then implemented. The ideas of kernel modules, makefiles and kernel memory allocation are then discussed. Finally, the meltdown attack is demonstrated and techniques to improve its efficacy are explored.

3.1 Measuring Memory Access Time

A very high level of accuracy is required when measuring time taken for memory access. The general paradigm for this involves reading CPU wall times before and after the access and measuring the difference between them. There is a large variety of APIs for getting the CPU wall time. For the system, the API used is rdtscp.

3.2 Flush+Reload

An oracle buffer of 256 entries is created. The first step is the flush operation. All the entries of the oracle are initialized and then flushed from the cache. This ensures that none of the oracle entries are cached beforehand, and prevents incorrect

results. Then execution of victim code is done. The victim uses a secret character (local to the victim function) to modify an entry in the oracle. When this happens the oracle entry gets cached. Next is to do a reload. Time taken to access each entry in the oracle is measured by the attacker's function. The entry that has minimum access time is the one cached by the victim. Hence, the index of the cached entry is obtained, which can then be used to get the secret value.

3.3 Meltdown

The attack targets kernel space memory from a user-space process. Two popularly used APIs for kernel memory allocation are vmalloc and kmalloc. Kmalloc guarantees that memory allocated is physically contiguous and offers higher performance. Vmalloc guarantees that memory allocated is virtually contiguous, and can distribute blocks if insufficient contiguous physical memory is available. Kmalloc is selected for the system since the attack is time sensitive. Kernel modules are utilized to allocate kernel space memory. For exception handling, the C signal library is used to catch segmentation fault and prevent fatal crash.

3.5 Delaying Commit – Keeping the ALU busy

As most modern speculative processors follow pipelined architecture where execution of an instruction is completed in four stages, Issue, Execute, Write-Result and commit. The commit phase occurs in order while execution phase occurs out of order. The exception generated by illegal memory access is thrown by the CPU only after the instruction has been committed. However, the attacking instructions can be executed out of order and the data can be stolen from the cache. To this end, the commit of the segmentation fault needs to be delayed as much as possible in order to improve the chances of the attack.

```
asm volatile(
    ".rept 1000;"
    "sqrtpd %%xmm0,%%xmm0;"
    ".endr;"

    :
    :
    :"eax"
);

kernel_data = *(char*)kernel_addr;
array[kernel_data*4096 + DELTA] +=10;
```

Figure 3.1: Arithmetically intensive assembly code preceding illegal memory access instruction

As shown in Figure 3.3, computationally intensive arithmetic instructions, such as square root, log calculation, etc. are inserted prior to the illegal memory access. This introduces a delay equal to the execution time of the ALU.

4. RESULTS

4.1 Flush+Reload

For proving the correctness, the Flush+Reload attack is run once with a secret key of 95. As shown in Figure 4.1 the leakage of the data occurs.

```
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
[redacted]/MyImpl$ ./FlushReload
Secret key = 236
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
[redacted]/MyImpl$ ./FlushReload
Secret key = 95
```

Figure 4.1: Snapshot of Flush+Reload

4.2 Meltdown

The attack is run on an Intel Haswell based 32-bit Virtual Machine running Ubuntu. Kernel module dynamically creates string "abcd" stored at a random location in kernel-space memory. It also creates a procfs entry of the secret key. To verify if kernel memory has been allocated, the module logs the base data address, the base data, and the string. The log can then be checked for the data as shown in Figure 4.2.

```
Dec 23 03:51:22 VM kernel: [11490.802372] Secret data a
ddress = f8735000
Dec 23 03:51:22 VM kernel: [11490.802375] Value at f873
5000 = abcd
Dec 23 03:51:22 VM kernel: [11490.802376] Data[0] = a
```

Figure 4.2: Log generated by the kernel module

To further ensure that the secret has been created and stored in the memory, the procfs is checked for the secret key file as shown in Figure 4.3.

```
[12/23/19]seed@VM:~/.../MyImpl$ ls /proc | grep secret
my_secret_key
[12/23/19]seed@VM:~/.../MyImpl$
```

Figure 4.3: Verifying the creation of procfs entry

The execution of the attack is then carried out targeting the kernel string as shown in Figure 4.4.

```
[redacted]:~/.../MyImpl$ ./meltdown
Success Rate = 57.800000%
[redacted]:~/.../MyImpl$ ./meltdown
Success Rate = 60.700000%
[redacted]:~/.../MyImpl$ ./meltdown
Success Rate = 76.700000%
[12/23/19]seed@VM:~/.../MyImpl$ ./meltdown
```

Figure 4.4: Running meltdown

5. MITIGATION

Only a redesign of the CPU architecture and new firmware is a permanent fix to the root cause of the problem, i.e., processor leaking out the value of protected memory locations via the cache. Apart from this, several mitigations are proposed as stop-gap measures to prevent the exploitation of meltdown on vulnerable CPUs. Major feature of Linux that was enabled with recent patches called KPTI [7, 8, 9, 10, 11]. The kernel maintains two separate page tables, one for kernel space processes that have full access to the memory, and another for user space processes, which have access only to their own memory. Another option is the microcode updates from Intel. Microcode is very low-level code that controls the operation of the CPU and is permanently embedded in the hardware. Intel [12] and ARM [13], have been rolling out microcode updates to patch the vulnerability via the Windows Update, and the Linux Kernel update distribution mechanisms. However, in some cases, these mitigations have resulted in a significant reduction in performance in both synthetic and real workloads [14].

5 CONCLUSION

A practical approach to exploiting meltdown on x86 based systems has been presented. It is observed that the CPU leaks the privileged data through the cache which exposes a timing channel that is exploited to steal the data. The success rate of the attack is significantly high. Proposed mitigations are found to be extremely effective. However, these mitigations have a significant performance impact in real and synthetic workloads. It is noted that the system has included a 4th generation Intel Haswell i5 CPU, which is relatively outdated. From the 8th generation coffee lake chips, Intel has altered the hardware design which makes it trickier to exploit this flaw. However, for a complete in-principle mitigation of the flaw, the hardware architecture of the CPU, including the wiring of cache and the boundary check control logic will need to be revisited by the manufacturers.

REFERENCES

[1] Lipp et. Al, "Meltdown: Reading kernel memory from user space". in Proceedings of the *27th USENIX Conf on Security Symposium*, 2018, pp.973-990.

[2] Yarom, Falkner, "Flush+Reload: A High resolution, low noise, l3 cache side channel attack," in *SEC'14 Proceedings of the 23rd USENIX Conf on Security Symposium*, 2014, pp.719-732.

[3] HUND, R., WILLEMS, C.,ANDHOLZ, T. "Practical Timing Side Channel Attacks against Kernel Space ASLR", in *S&P*, 2013.

[4] Taylor Hornby, "*Side-channel attacks on everyday applications: distinguishing inputs with FLUSH+RELOAD".* Available: https://www.blackhat.com/docs/us-16/materials/us-16-Homby-Side-Channel-Attacks-On-Everyday-Applications-wp.pdf.

[5] Kocher et. Al, "Spectre attacks: Exploiting speculative execution". Proceedings of the *40th IEEE Symposium on Security and Privacy*, 2019.

[6] TOMASULO, R. M, "*An efficient algorithm for exploiting multiple arithmetic units*". **IBM** Journal of research and Development, 1967.

[7] HANSEN, D. [PATCH 00/23] KAISER: *unmap most of the kernel from user space page* tables, Available: https://lkml.org/lkml/2017/10/31/884, Oct 2017.

[8] HANSEN, D. [v2] KAISER*: unmap most of the kernel from user space page tables*, Available: https://lkml.org/lkml/2017/11/8/752, Nov 2017.

[9] HANSEN, D. [v3] KAISER: *unmap most of the kernel from user space page tables*, Available: https://lkml.org/lkml/2017/11/10/433, Nov 2017.

[10] HANSEN, D. [v4] KAISER: *unmap most of the kernel from user space page tables*, Available: https://lkml.org/lkml/2017/11/22/956, Nov 2017.

[11] LWN, *The current state of kernel page-table isolation*, Available: https://lwn.net/SubscriberLink/741878/eb6c9d3913d7cb2b/, Dec. 2017.

[12] INTEL, *Intel analysis of speculative execution side channels*, Available: https://newsroom.intel.com/wp-content/uploads/sites/11/2018/01/Intel-Analysis-of-Speculative-Execution-Side-Channels.pdf , Jan. 2018.

[13] ARM, *Software implications for Spectre/ Meltdown on Arm cores, Version 1.3*, Available: **Error! Hyperlink reference not valid.**,Jun. 2018.

[14] A. Prout et al, "Measuring the impact of spectre and meltdown" in *High Perf Extreme Computing Conf. IEEE*, 2018, pp. 1–5.

[15] INTEL, *IA-PC HPET Specification Rev 1.0a 1 IA-PC HPET (High Precision Event Timers) Specification*. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technical-specifications/software-developers-hpet-spec-1-0a.pdf

[16] UEFI, *Advanced Configuration and Power Interface Specification*, Available: https://uefi.org/sites/default/files/resources/ACPI_6_2.pdf

[17] AMD, *Game Timing and Multicore Processors*, Available:http://developer.amd.com/wordpress/media/2013/03/Game_Timing_Multicore_Processors.pdf

[18] FAULKNER, GOMES, "The Process file system and process model in unix system V". Proceedings of *the USENIX Conf*, 1991, pp.243-252.

[19] JOHNSON, K. KVA *Shadow: Mitigating Meltdown on Windows*, Available:https://blogs.technet.microsoft.com/srd/2018/03/23/kva-shadow-mitigating-meltdown-on-windows/, March 2018.

[20] Zhichao Hua , Dong Du , Yubin Xia , Haibo Chen , Binyu Zang, EPTI: "efficient defence against meltdown attack for unpatched VMs". Proceedings of the *2018 USENIX Conf*, July 11-13, 2018, Boston, MA, USA.

[21] PHORONIX, *Linux 4.12 To Enable KASLR By Default*, Available:https://www.phoronix.com/scan.php?page=news_item&px=KASLR-Default-Linux-4.122017

[22] HENNESSY, J. L., AND PATTERSON, D. A. *Computer Architecture: A Quantitative Approach*, 6 ed. Morgan Kaufmann, 2017.

[23] Greg Kroah-Hartman, *Linux 4.15.1*, Available: https://cdn.kernel.org/pub/linux/kernel/v4.x/ChangeLog-4.15.1

[24] Linux Kernel Foundation**.** *Memory Allocation Guide* Available: https://www.kernel.org/doc/html/latest/core-api/memory-allocation.html