POSTER: High Efficiency, Low-noise Meltdown Attack by using a Return Stack Buffer

TaeHyun Kim
Kwangwoon University
Seoul, Republic of Korea
taehyun9203@gmail.com

Youngjoo Shin
Kwangwoon University
Seoul, Republic of Korea
yjshin@kw.ac.kr

ABSTRACT

Meltdown attack exploits out-of-order execution in modern microarchitectures to extract sensitive data in kernel space of operating systems. Out-of-order execution opens a window of transient execution in which unauthorized access to kernel space is allowed. The original Meltdown attack utilizes an OS signal handler and hardware transactional memory support (e.g., Intel TSX) to create transient executions. Both methods, however, restrict the effectiveness of the attack due to a large amount of system noise from signal handlers and a limited number of processors that support TSX. To overcome this limitation, we propose a new variant of Meltdown attack by using a return stack buffer (RSB). Without the aid of TSX, the proposed attack introduces lower level of noise than the signal handler-based method, which broadens the impact of Meltdown attacks to a wide range of processors. We conclude this paper by presenting several countermeasures against the proposed attack.

KEYWORDS

Microarchitectural side-channel attacks, Transient execution attacks, Return stack buffer

ACM Reference Format:
TaeHyun Kim and Youngjoo Shin. 2019. POSTER: High Efficiency, Low-noise Meltdown Attack, by using a Return Stack Buffer. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3321705.3331009

1 INTRODUCTION

Meltdown attack [1, 5] introduces serious security threats to a various computing environments ranging from a PC to cloud computing. This attack utilizes CPU optimization techniques such as out-of-order execution to extract sensitive information from a victim.

The original Meltdown attack utilizes a OS signal handler and hardware transactional memory support (e.g., Intel TSX) to create transient executions. Signal handler generally handles exception through the support of operating systems. Because exception should be handled in kernel mode, it is necessary to change mode from user mode to kernel mode. After exception handling completed, the attack code goes back to user mode and continues to execute next instructions where exception occurred. This procedure spends long time to complete the attack, which introduces a large amount of system noise to cache covert channel in Meltdown attacks.

Alternative to the signal handler-based method is to utilize Intel TSX, a hardware transactional memory support. TSX suppresses exception using transactional memory. More specifically, a user-controlled code running in TSX context handles the exception itself instead of passing it to a OS signal handler. Owing to the exception suppressing, TSX allows fast and low-noise cache covert channel in Metldown attacks. However, the TSX-based approach is very restrictive because there are a narrow range of TSX-enabled processors.

In this paper, we propose a new variant of Meltdown that improves the attack performance. Our attack utilizes a return stack buffer (RSB), which is a kind of speculative execution units equipped in most processors. RSB enables low-noise and high efficient cache covert channel without the aid of the hardware transactional memory support. Thus it broadens the impact of Meltdown to a wide range of processors. We also present several possible countermeasures against the proposed attack.

The rest of this paper is organized as follows. In Section 2, we give some background knowledge about our attack. In Section 3, we describe the proposed attack and its building blocks in detail. In Section 4, we describe our two experiments to show our attack is effective. In Section 5 and 6, we present several countermeasure against our attack and related work, respectively. Finally in Section 7, we conclude this paper by summarizing our work.

2 BACKGROUND

2.1 Return Stack Buffer

When a processor executes a call instruction, it takes long time to calculate a return address for the call instruction. To reduce the computation time and the overhead by misprediction of return addresses, CPU vendors made a special hardware buffer used only for return addresses. This buffer is called a return stack buffer (RSB). It usually consists of 16 - 32 entries, which is dependent on processors. RSB stores return addresses in a stack when call instruction is called. Then the stack pointer points a top of stack that has return address. When a return instruction is executed, the processor pops the top of the stack buffer and then branches to the return address.

2.2 Cache covert channel

Cache covert channel is an application of cache side channel attacks. Flush+Reload [3, 7] is one of the cache side channel attacks. It

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07.
https://doi.org/10.1145/3321705.3331009

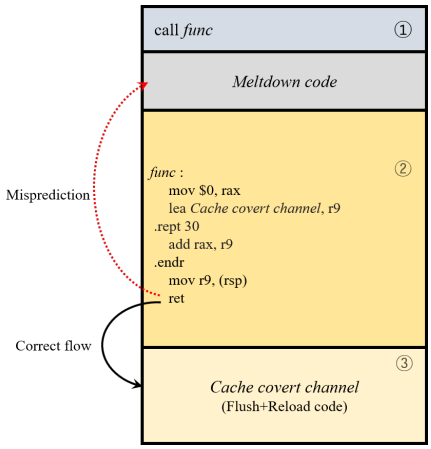

Figure 1: Building block of our attack

enables fine-grained attack to extract secret value without intervention of a victim. This attack needs shared memory between victim and spy, which means the attack and the victim share some cache lines. In this attack, the spy first executes clflush instruction to flush shared cache line. After waiting for a while, the spy reloads the flushed cache line again. If the victim used the cache line, reloading time about flushed cache line is short but on the opposite case, reloading time is long. By exploiting the time difference, the spy knows shared cache line using victim or send spy secret value in cache line.

3 ATTACK DESCRIPTION

Our attack is based on Meltdown attack [5]. The method used in the original attack exploits a signal handler to deal with page fault exception. However, the signal handler-based method needs context switching from user to kernel mode, which incurs high level of system noise in cache covert channel where information is extracted. Another approach proposed in the original Meltdown attack utilizes Intel TSX, a hardware transactional memory support. Thanks to the ability of suppressing page fault exception, TSX enables highly efficient cache covert channel with low noise. The main drawback, however, is its limited applicability due to small range of TSX-enabled processors.

In this paper, we propose a new variant of Meltdown-type attacks that overcomes the aforementioned limitations. Our method is to utilize a return stack buffer (RSB), which is a kind of speculative execution units equipped in most processors.

The overall procedure of the proposed attack is depicted in Fig.1. Our attack procedure consists of three steps. First, the attack code calls a function named *func* (See ① in Fig.1). This function consists of two parts: (a) repetitive add instructions with an EAX register, and (b) a mov instruction that modifies return address in stack followed by a ret instruction. The part (a) has a purpose of introducing a long dependency chain of instructions to delay the retire of the ret instruction in part (b).

As a second step of the procedure, the function *func* is executed (See ② in Fig.1). More specifically, a return address of the function *func* is replaced with a location of execution code that extracts information through cache covert channel. After that, the processor executes a ret instruction. Because of long data dependency introduced in part (a), it takes time to determine the correct destination address. Thus, the processor attempts to predict a destination of the branch (i.e., return address) by consulting a RSB for possible destination. The RSB caches pairs of a call instruction and its corresponding return address, which refers to the location next after the call instruction. In our case, it is the location where the actual Meltdown code exists. As a result, the processor jumps to the location and speculatively executes a sequence of instructions that makes unauthorized access to kernel. The speculation will eventually fail since a return address has been previously modified in part (b). Hence, the processor cancels the speculative execution and discards its result. As the original Meltdown attack, however, the micro-architectural state (i.e., cache state) still remains changed by the result even after the execution was discarded.

In the third step, the processor corrects the destination and jumps to the location at the changed return address, where measurement code (i.e., Cache covert channel) is located (See ③ in Fig.1). This code measures the cache state by using a Flush+Reload technique. The measurement results are then decoded and secret information in the specified kernel address is finally extracted.

Environment	CPU Model	Cores	Operating System
Notebook(Lenovo ThinkPad X220)	Intel(R) Core(TM) i5-2540M CPU @ 2.60GHz	2	Ubuntu 14.04 Server
Desktop	Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz	4	Ubuntu 14.04 Server
Desktop	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz	4	Ubuntu 14.04 Server

Figure 2: Experimental environment

4 EXPERIMENT

In order to validate the effectiveness of our attack, we conducted some experiments. First, we performed an experiment to measure the performance in breaking Kernel address space layout randomization (KASLR). We also performed another experiment to measure the performance in extracting secret strings in physical memory. The experimental environment consists of three different processors (See Fig.2).

4.1 Breaking KASLR

KASLR is a security mechanism of modern operating systems to protect kernel space. By randomizing kernel address space, it prohibits attackers from compromising the kernel. KASLR is known to be vulnerable to Meltdown-type attacks [5]. The attacker is able to de-randomize kernel address and extract direct physical map offset in kernel area.

We conducted an experiment to show the efficiency of the proposed attack in breaking KASLR. The experimental result is presented in Fig.3. The graph in the figure shows the number of successful extractions of direct physical map offset in a minute. In order to examine the attack performance in various conditions, we used stress-ng[1], a tool for system stress testing. Since the attack performance is influenced by cache and CPU activities, we run stress-ng in background with only two options enabled, which make CPU and cache busy. The experiment shows that our attack that utilizes

[1]https://kernel.ubuntu.com/ cking/stress-ng/

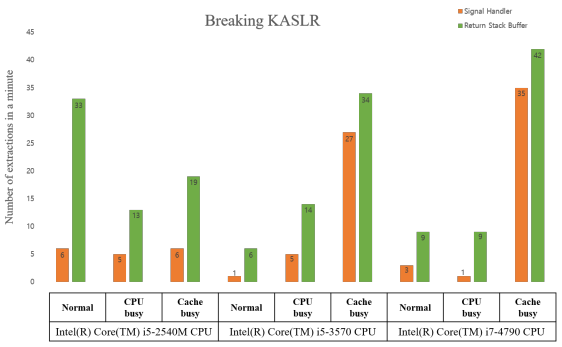

Figure 3: Number of extractions of direct physical map offset

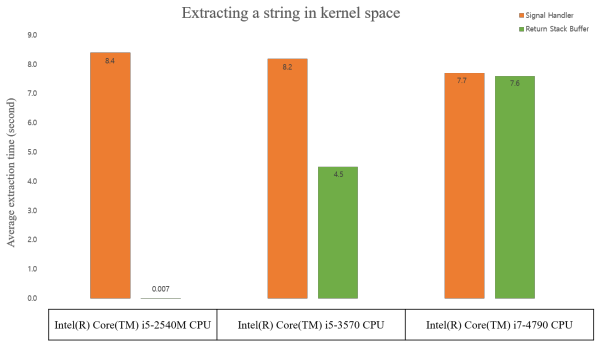

Figure 4: Average time of extracting a string in kernel space

a RSB has higher efficiency than signal handler-based method used in the original Meltdown attack in various conditions.

4.2 Extracting a string in kernel area

We conducted another experiment to measure the attack performance in extracting a (secret) string in kernel area. In this experiment, we assume that a virtual address of the target string is known to attackers. In fact, the attackers can easily determine a virtual address of the target string by using direct physical map offset. That is, the virtual address of the string can be calculated by adding the direct physical map offset with a physical address of the string.

For ease in conducting the experiment, we ran two processes. The first process runs in supervisor mode and returns a physical address of the target string using pagemap. Given two arguments of direct physical map offset and physical address returned by the first process, the second process actually attempts to extract the string. The experimental result is shown in Fig. 4. The graph in the figure shows average elapsed time of successful extractions of the secret string in physical memory. This result shows that the RSB-based method is more efficient than the signal handler-based method in extracting strings in kernel area.

5 COUNTERMEASURE

There are some countermeasures against the proposed attack. One of these is to isolate a kernel page table from user space, which is known as KAISER [2]. This technique divides address space into user address space and kernel address space using CR3 register. So, this prevents attackers in user mode who attempt to break KASLR. The another method is to refill RSB on every context switching. In fact, this technique has been already introduced since Skylake processor. Hence, these processors are not affected by the proposed attack.

6 RELATED WORK

The original Meltdown attack [5] utilizes a signal handler and hardware transactional memory support, both of which are very restrictive in practical scenario. Other work known as ret2spec [6] and spectre return [4] exploit RSB in introducing new variants of Spectre attack. Although both are similar to the method proposed in this paper, our work is mainly focused on showing the benefits of RSB over a signal handler or TSX in mounting Meltdown-type attacks.

7 CONCLUSION

In this paper, we proposed a new Meltdown attack using a return stack buffer. The proposed method exploits out-of-order execution and speculative execution in RSB. To validate the effectiveness of our attack, we conducted several experiments with various conditions. We also presented some countermeasures such as KAISER and RSB refilling to mitigate our attack. Our study showed that a return stack buffer can be used as a method to efficiently extract secret.

ACKNOWLEDGEMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2019-0-00533, Research on CPU vulnerability detection and validation)

REFERENCES

[1] Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin Von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, and Daniel Gruss. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. *arXiv.org* arXiv:1811 (2019).

[2] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, ClÃĬmentine Maurice, and Stefan Mangard. 2017. KASLR is dead: Long live KASLR. In *International Symposium on Engineering Secure Software and Systems (ESSoS 2017)*, Vol. 10379 LNCS. 161–176.

[3] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In *Proceedings of the 24th USENIX Security Symposium*. 897–912.

[4] Esmaeil Mohammadian Koruyeh, Khaled Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *USENIX WOOT 2018*.

[5] 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 Security Symposium*.

[6] Giorgi Maisuradze and Christian Rossow. 2018. ret2spec: Speculative Execution Using Return Stack Buffers. In *Proceedings of the ACM conference on Computer and communications security - CCS '18*.

[7] Yuval Yarom and Katrina Falkner. 2014. Flush + Reload : a High Resolution , Low Noise, L3 Cache Side-Channel Attack. In *Proceedings of the 23th USENIX Security Symposium*. 719–732.