SCIENCE CHINA
Information Sciences

• LETTER •

September 2021, Vol. 64 199104:1–199104:3
https://doi.org/10.1007/s11432-018-9678-8

On-demand cut off the covert channel to mitigate meltdown

Yusong TAN[1], Baozi CHEN[1*], Liehuang ZHU[2], Qingbo WU[1], Peng ZOU[1] & Yuanzhang LI[2]

[1]*College of Computer Science, National University of Defense Technology, Changsha* 410073*, China;*
[2]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing* 100081*, China*

Received 13 September 2018/Accepted 28 November 2018/Published online 27 May 2021

Citation Tan Y S, Chen B Z, Zhu L H, et al. On-demand cut off the covert channel to mitigate meltdown. Sci China Inf Sci, 2021, 64(9): 199104, https://doi.org/10.1007/s11432-018-9678-8

Dear editor,

Recently, researchers have discovered a new kind of attack named Meltdown [1] that exploits the out-of-order execution behavior of modern processors and side-channels. The adversary can use this method to overcome the memory isolation that is protected by a processor and read the entire kernel memory of a machine it executes on. Mitigating this new attack is thought to be expensive at the moment because current solutions are based on common optimization techniques for modern processors.

Mitigating Meltdown can be achieved through either hardware redesign or software workaround. It is a straightforward idea to fix the issue by enabling the hardware to identify whether a memory fetch would violate a security boundary and prevent execution. Since hardware redesign is expensive and not applicable to existing systems, software workaround is the best short-time solution at the moment. According to Lipp et al. [1], KAISER (kernel address isolation to have side channels efficiently removed) [2] can be used as a software countermeasure against Meltdown that modifies the operating system to prevent the kernel from being mapped in the user space. In the worst case scenario, it would introduce about 30% regression on a loop-back networking test [3]. We present a new method that inserts a gadget on the path to exception handlers in the Linux kernel. Any application that is not on the white list and raises an exception may trigger the gadget to flush the cache lines so that the side-channel would be cut off. Since an exception is rarely raised, it has little impact on the performance of normal applications.

To perform a Meltdown attack, the adversary would repeat the following three steps. First, an inaccessible memory address or privileged system registers are loaded into a register. Second, a transient instruction performs memory accesses to allocate cache lines based on the register that has been loaded with sensitive contents. Third, the attacker uses Flush+Reload [4] to extract the sensitive information encoded in the timing channel of the memory locations accessed in the second step. During the first step, the access to privileged resources from a user space program would finally raise an exception because its permission level does not allow it to do so. However, under out-of-order execution, there are a number of instructions in a small time window before the exception is finally raised. Hence, a race condition occurs. The attacker can then fill this time window with malicious codes that allocate cache lines to record sensitive contents in a microarchitectural state. After the exception is raised, the pipeline would be flushed. All the changes in the architectural state made by the malicious transient instruction sequence would be rolled back, but the microarchitectural side effects will be retained. The exception would then generally terminate the user space program. Thus, an attacker can exploit this to extract sensitive content from the covert channel.

Since exceptions would be triggered when performing a Meltdown attack, we propose a method that injects noise into the covert channel or resets the microarchitectural states when an exception is detected. By injecting noise into the covert channel, an attacker could extract some information from the channel but incorrectly. Otherwise, the operating system would explicitly flush the cache lines, eliminating the footprint that the transient instructions recorded in the microarchitectural states. Figure 1 shows the program flow after inserting a gadget on the path to cut off the covert channels to exception handlers. As long as an exception is raised, the source instruction has to be retired, in which stage the out-of-order architecture executes in order. Therefore, no transient instructions would be executed after an exception is raised. Hence, the noise or the cleanup states would be maintained. No further race condition would occur between the transient instructions and the gadget before the adversary tries to extract the contents in the side-channel. As the exception is triggered only on the path to exception handlers, as it will have less impact on the performance of most applications. Further, it can be used to mitigate rogue data cache load as well as rogue system register read which cannot be achieved using kernel page-table isolation

* Corresponding author (email: chenbaozi10@nudt.edu.cn)

(KPTI). We verified this with the proof-of-concept program published on Github [5].

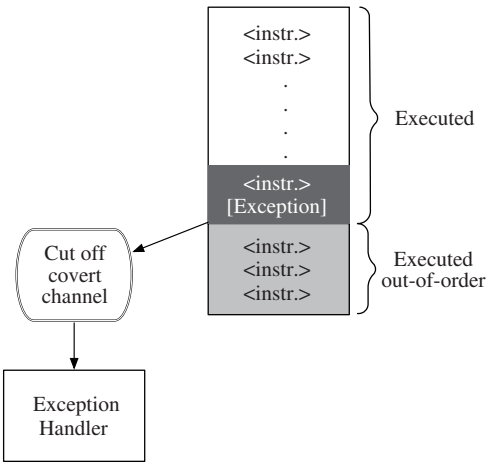

Figure 1 Inject noise or sweep up microarchitectural states when the Meltdown attack triggers an exception.

We implemented the mechanism in a mainline Linux kernel v4.15 on the ARM platform and ran several benchmarks to evaluate the impact of our new approach on performance compared to KPTI. Under these two kernels, we launched various tests targeting performance, from the kernel to the user space. We first ran UnixBench to get a general view of the system performance. Then, we ran SPECCPU2006 under those two kernels to survey the impact of the kernel mitigation on pure computing in user space. For IO (input/output) performance, dbench was chosen to evaluate the storage workloads while tbench was used to measure the network stack. The results show that our method has little impact on the whole system, whereas KPTI impacts performance in some scenarios. The overhead introduced by the KPTI mainly reflects in the results of UnixBench and tbench that represent the general view of the system and network stack IO performance, respectively. Our method gains about 15% more scores in UnixBench and over 16% more throughput in tbench tests. Both mitigation schemes have little impact on the overall computing performance for the system as well as the disk IO.

The on-demand cut off method can mitigate Meltdown with fewer overheads than KPTI because it avoids unmapping kernel address while switching to the user space that will also trigger the processor to flush the TLB (translation look- aside buffers). The key feature we utilize is the exception that will be raised when the adversary accesses privileged resources from the user space. In the original Meltdown study, the authors proposed two approaches for handling exceptions: exception handling and exception suppression. To handle exceptions, an attacker may choose to folk a child process to execute transient instruction and be killed by the signal or simply install a signal handler to replace the default termination operation. This can easily be caught because the exception is triggered explicitly. However, mitigating attacks that suppress exceptions would involve more complexity.

On the Intel platform, the attacker can take advantage of transactional synchronization extensions (TSX) to suppress exceptions. With TSX technology, the instruction that breaks the privilege level can be wrapped into a transactional code region that starts with `XBEGIN` and ends with `XEND`. Once the instruction violates the permission, the transaction failure would redirect the CPU to roll back the architectural state and continue to execute rather than raise an exception. Fortunately, TSX has its own exception mechanism, with which we can configure PMU (performance monitor unit) counters to raise an interrupt once a specific number of RTM (restricted transactional memory) abort is triggered. On a platform without hardware transaction memory such as TSX, the exception can also be suppressed by exploiting speculative execution of a mispredicted branch similar to Spectre attack [6]. To train the branch predictor, the attacker can make the CPU execute only transient instructions speculatively by ensuring that the branch condition is always false. To deal with this situation, we can use the method introduced by Pierce et al. [7] which utilizes a PMU counter overflow event to raise an exception on the mispredicted branch.

With the on-demand cut off mechanism, it is difficult to extract sensitive content through the cache timing side-channel, but the attacker can use the malicious Meltdown code to slow down the system performance and launch a DoS (denial-of-service) attack. Hence, a mechanism is necessary to detect whether there is a real Meltdown attack and terminate the malicious process accordingly. A counter could be used to record the number of times that the gadget is executed. When it exceeds a preset threshold, a scanner is triggered to check whether there are malicious codes in the instruction window. If no malicious codes are detected or the program is in the white list, then either the gadget or the detection is turned off and the entire system continues to run as usual. Otherwise, the attacker is exposed, and the malicious process will be killed. Recalling that out-of-order execution is a restricted form of data flow computation [8], all the malicious transient instructions in a Meltdown attack have to be within an instruction window that begins with an instruction that violates the permission level. Therefore, the number of instructions that need to be scanned is limited, thereby introducing little overhead.

Conclusion. In this study, we present a new method for mitigating Meltdown attacks that cut off the side-channel on-demand when an attack raises an exception. We inserted a gadget on the path to exception handlers in the Linux kernel. If any application that is not on the white list raises an exception, it may trigger the gadget to flush the cache lines to cut off the side-channel. Since an exception is rarely raised, it has little impact on the performance of normal applications. It eliminates the overhead introduced by KAISER while mitigating the rogue system register read variant. We implemented a prototype on Linux kernel v4.15 and evaluated the performance overhead. The experimental result shows that it gains about 115% in performance scores as compared to KAISER in UnixBench.

Acknowledgements This work was supported by National S&T Major Project of China (Grant No. 2016ZX01040101) and National Natural Science Foundation of China (Grant Nos. 61602498, 61872444).

References

1 Lipp M, Schwarz M, Gruss D, et al. Meltdown: reading kernel memory from user space. In: Proceedings of the 27th USENIX Security Symposium, 2018. 973–990
2 Gruss D, Lipp M, Schwarz M, et al. KASLR is dead: long live KASLR. In: Proceedings of International Symposium on Engineering Secure Software and Systems, 2017. 161–176

3 Hansen D. KAISER: unmap most of the kernel from userspace page tables. https://lkml.org/lkml/2017/11/22/956
4 Yarom Y, Falkner K. FLUSH+RELOAD — a high resolution, low noise, L3 cache side-channel attack. In: Proceedings of USENIX Security Symposium, 2014. 719–732
5 Gorgovan C. Dump privileged ARM system registers from usermode using variant 3a of Meltdown. https://github.com/lgeek/spec_poc_arm
6 Kocher P, Genkin D, Gruss D, et al. Spectre attacks: exploiting speculative execution. 2018. ArXiv: 1801.01203
7 Pierce C, Spisak M, Fitch K. Capturing 0day exploits with perfectly placed hardware traps. In: Proceedings of the Black Hat USA, 2016
8 Patt Y N, Hwu W W, Shebanow M C, et al. HPS, a new microarchitecture: rationale and introduction. In: Proceedings of International Symposium on Microarchitecture, 1985. 103–108