2018 15th International Conference on
Control, Automation, Robotics and Vision (ICARCV)
Singapore, November 18-21, 2018

Multi-Agent Cross-Platform Detection of Meltdown and Spectre Attacks

Xinxing Zhao, Chandra S.Veerappan, Peter. K.K Loh, Zhaohui Tang and Forest Tan

Abstract— **Modern processors use out-of-order and speculative execution to maximize performance. However, these types of optimization executions may leak users' confidential information to another malicious party through a side channel, as can be seen from the notorious Meltdown and Spectre PoC attacks. Although some big companies have released software patches and updates to work around the hardware problems, however, they might inadvertently cause some stability or performance issues, especially for Windows systems. Furthermore, some legacy systems are never going to be patched, such as XP systems. Therefore, we provide a more proactive method, a multi-agent way of detecting Meltdown and Spectre on Windows (including the XP System), Linux and OS X Systems. Experiments show that our detection mechanism is very effective in detecting these attacks for all the major existing operating systems.**

I. INTRODUCTION

Speculative execution is an optimization technique that is employed in modern high-speed processors to increase performance. The technique allows a computer system to perform in advance these tasks that may not be needed, by guessing future execution paths and executing the instructions in them in advance. This way some extra work is completed before it is known whether it is actually needed, so the delay that would have to be incurred may be prevented. If the guess was correct, then the computational results in the extra work are committed, producing a bonus performance gain. However, if it turns out that the execution paths were wrong, and the work was not needed in the end, most changes made by the executions in advance are discarded and the results are ignored, the states of registers are reverted back to their original conditions. From a security point of view, speculative execution may cause a program to run in incorrect ways. However, as processors can return back to its original saved checkpoint state to maintain correctness, these incorrect ways of execution were thought to have no security implications previously.

Out-of-order optimization technique is an important feature used in most of today's high-performance central processing units, which allows processors to make the maximum of usage of instruction cycles that would otherwise be wasted. With the technique, instructions in a processor are executed in such an order that can be dictated by the availability of input data and execution units, therefore the original order in a program and actual order that has been executed might be very different. In this way, the processor can process ahead with the subsequent instructions that can be run immediately and independently instead of being idle there and waiting for the preceding instruction to be completed, so long as the results follow the architectural definition. From the security point of view again, out-of-order optimization technique may lead an unprivileged process to load data from a privileged address into a temporary CPU register. What makes this worse is that, the register value can be referenced by the CPU to do some further computations. As processors are designed to ensure the correctness of executing any program and discarding the memory lookups if the work was not needed, this results in performance comparable to idling. At the architectural level, this out-of-order execution feature, was also assumed to have no security implications previously.

Security implications, however, turns out to be very real and serious with the revelation of the notorious Meltdown and Spectre [1] attacks in early 2018. Although these vulnerability exploits may still just be working proofs of concept (PoC) or reportedly experimental for now, it's only a matter of time before they will be fully weaponized. Meltdown affects desktops, laptops, and cloud computers with the out-of-order execution feature built as early as 1995; While almost all the desktops, laptops, cloud servers, as well as some advanced smartphones are affected by Spectre.

The exploitable vulnerabilities exist fundamentally at the hardware level; however, big companies are rolling out software patches to cope with the issues. The KAISER patch [2], developed originally to prevent side-channel attack targeting Kernel Address Space Layout Randomization (KASLR), coincidently preventing Meltdown attacks in Linux Systems. Intel, Microsoft, Apple, and Google, all had their patches released and more are on the way. But many problems were also caused as a result of these patches. Some PC users reported that their system (Windows 10, Windows 8.1 and Windows 7) failed to boot following the installation of the security patch [21]; Some data centers reported that performance degraded by 18% to 25%; Approximately 2% to 4% performance degradation is reported on other systems [22]. In the end, Intel backtracked and decided to advise customers not to download the patches.

Apart from the possibly problematic patches for the systems, it's also important to establish more proactive strategies in hunting, detecting, and responding to these threats. We worked on and developed a multi-agent system called CELLS [3], which has a detection function for attacks that exploit Meltdown and Spectre by utilizing statistical

*Research supported by Singapore Ministry of Education, MOE2016_TIF-1-G-022.

The Authors are all with Singapore Institute of Technology, Dover, 138583 Singapore {xinxing.zhao, chandra.veerappan, peter.loh, zhaohui.tang, forest.tan}@ Singaporetech.edu.sg

performance monitoring. The multi-agent design enables each micro-agent to be targeted towards a specific family of attacks with a small footprint, facilitating deployment on resource-constrained devices. It works cross-platform, as we used it successfully in detecting Spectre and Meltdown PoC attacks in Linux, Windows and OS X platforms. This detection function or mechanism can be used as an alternative or a complement to the patches, especially for those systems whose patches may cause stability or performance issues. It is also worth noting that, older systems, e.g. Windows XP systems, a lot people are still using, most probably will never be patched, and our detection mechanism can also be applied to protect them.

II. Background

A. CELLS

The current work continues from our previous research [3,4]. Our main objective is to develop a light-weight, distributed, cross-platform and targeted software agent (called CELLS), that can be used to monitor and detect existing cyber-attacks, not only in systems such as Linux, Windows, Mac, but also for Android and Raspberry Pi systems. There are some existing cross-platform frameworks and technologies that we can choose from, such as ZeroMQ and QT. It is worth noting that neither Meltdown nor Spectre will affect Raspberry Pi, as well as some low-end Android devices, due to absence of speculative capabilities in their processors [5]. By using lightweight multi-agent solution through CELLS, agents can move across the different computing nodes under users' instructions and detect different attacks. Different agents can also communicate and collaborate across different target systems automatically with TCP sockets. Figure 1. shows the basic functions of the cells.

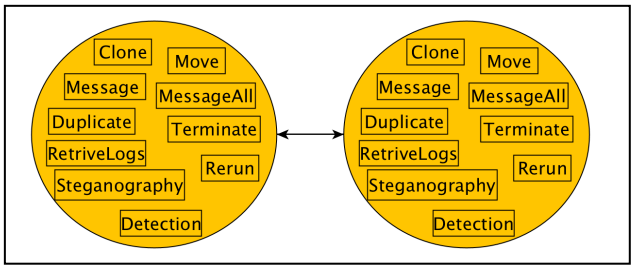

Figure 1. Cells Communications Sub-Systems

B. Cache Side-Channel Attack

Programs that are running on the same system may leak information to each other due to the memory page sharing and the hardware protection not being fully transparent [5]. In addition to page sharing, processes running on the same processor share the processor caches, and the Last-Level Cache (LLC) is shared among all the cores. Modern processors employ a cache hierarchy of successively smaller but faster caches to speed-up memory accesses and address translation. Getting data from memory or from cache levels closer to memory takes longer than getting it from cache levels closer to the core. Several types of side-channel attacks, such as Evict+Time [6], Prime+Probe [7], and Flush+Reload [8], has been based on this timing difference. Flush+Reload attacks, exploiting the shared, inclusive last-level cache, can identify accesses to specific memory lines (lines are fixed-size chunks divided from cache memory) in the cache. Therefore, they have a high fidelity. An attacker starts with flushing a targeted

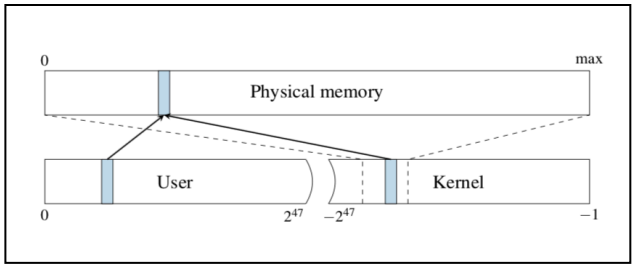

Figure 2. The memory mappings on Linux and OS X systems [1]

memory location (cache line shared with the victim) using the x86's *clflush* instruction and waiting there for the victim to access the cache line. After the victim executes for a while, the attacker reloads the memory line, measures the time it took for the reloading. During the waiting period, assume a victim accessed the monitored cache line, then the reloading and reading operation will be fast, because the data will be readily there in the cache. On the other hand, the reloading operation will be significantly slower, if the victim has not accessed the line during the waiting, as the line will need to be brought from memory. Therefore, an attacker can know which data is used during the waiting time, by measuring the timing difference it takes to reload the data from a cache line in a different process. The Flush+Reload attack has been used in various cases. [9,10,11,12]

C. Memory Mapping

As modern computers usually run in parallel many processes, the memory mapping (virtual paged memory) is used to fulfill the requirement of isolation of processes from each other, and to achieve the efficiency and controllability of the memory access.

Virtual address spaces are divided into a set of pages that can be individually mapped to physical memory through a multi-level page translation table. In order to enforce privilege controls (such as readable, writable, executable and user-accessible), an operating system defines protection properties in these translation tables. Any attempts that access authorized memory will succeed immediately, on the other hand an exception will be caused when trying to access an unauthorized memory. A special CPU register is holding the current translation table. On each context switch, the next process's translation table address will be updated to the register to implement per process virtual address spaces by the operating system. Therefore, each running process can keep their individual activity and integrity without interference with another (hence no information leak to another) as it can only reference data which belongs to its own virtual address space.

There are two parts in each virtual address space, namely a user part and a kernel part. A running application can access the user address space at any time, while only if the CPU is running in privileged mode can access the kernel part. The operating system disables accessible property of the corresponding translation table to enforce this rule. A computer system can refer far more memory than it will ever physically have with the virtual memory mechanism and it can be immensely sped up by "mapping" in effect all memory of all active processes into every process's virtual memory. This is the case for both Linux and OS X systems, whereas in the Windows system, although with a different mapping

mechanism, still a large portion of the in-use physical memory of a process is mapped into every other process' kernel part. Figure 2 shows how an address in physical memory is mapped into user space, and also mapped into the kernel. In order to mount memory corruption attack, the exploitation often requires the knowledge of addresses of specific data. Therefore, there are many ways to block this type of attack, such as address space layout randomization (ASLR) as well as non-executable heaps and stacks; KASLR is used to protect the kernel, resulting in more work to be done for an attacker as he needs to guess the location of kernel data structures. Unfortunately, there are still ways provided by side-channel attacks to find the exact location of kernel data structures [12, 13, 14] and to derandomize ASLR in JavaScript [15]. After getting the knowledge of these addresses and combining with certain software bug, a well-crafted attack can easily lead to privileged code execution.

D. *Spectre Attack*

Spectre attacks [1] are based on two main factors: 1. The speculative execution - more specifically, to manipulate branch prediction in modern microprocessors, induce the processor to speculatively executing instructions sequences that are not supposed to execute. 2. Cache side channel attacks- on most processors observable side effects may have left behind after the speculative executing of instructions (that are not supposed to execute), which in turn may reveal private information to attackers through a side channel. There are two Spectre variants, namely the Spectre-V1, bounds check bypass [16]; and the Spectre-V2, branch target injection [17], have been issued by CVE system. Intel had developed a new generation of hardware that has fixes for Meltdown and Spectre-V2, but not for Spectre-V1. It seems true that "As Spectre is not easy to fix, it will haunt us for a long time" [1]. Therefore, the spectre attack PoC code in the appendix of Spectre paper [1], which is a Spectre-V1 type, can be used as a good start for us to study and test for some time, and we will use this attack to test our detection mechanism in all three major operating systems (Windows, Linux, OS X).

E. *Meltdown Attack*

Meltdown [1] attacks are based on some widely used features: 1. The out-of-order execution – allows the instructions to run out of order and in parallel to maximizes efficiency and make the maximum usage of CPU cycles. 2. The memory mapping and privilege level control. As has been mentioned, the operating system enforces the rule of which processes are authorized and which are not authorized to access the areas of virtual memory. Suppose a memory operand is referenced when a CPU attempts to execute an instruction. The operand's address is calculated and the value at the address is read, however if this operand's address is prohibited from this process by the virtual memory system and privilege check, then the execution unit must discard the results of the memory read. 3. The cache side channel timing attacks. We continue the process above, although memory read was discarded by the execution unit. However, one of the side effects is the data caching, as the data at the forbidden address may have been read before the privilege check (out-of-order execution, instruction execution can be executed by the CPU before the privilege checking). If this were what actually had happened, the mere act of caching can cause a leak of information, as the process can execute instructions that reference memory operands directly and performing a timing difference attack, because referencing the same address directly will execute faster, therefore determining the forbidden address and the value therein. Meltdown uses these widely used features, and with the help of exception handling or exception suppression techniques, can read any and every address of interest at high speed. According to [1], they can read the kernel memory at up to 503 KB/s.

III. THE DETECTION

In this section, we describe the CELLS detection mechanism. We start with some interesting and important observations which formed the basis for the detection mechanism. We then present our results in detecting Meltdown and Spectre PoCs across different platforms, i.e., Windows, Linux and OS X.

A. *Our Detection Mechanism*

We use statistical performance monitoring to observe the system behavior. We observed that when a Spectre or a Meltdown attack occurs, some of measured parameters jump to levels that are as high as 100 times more than the non-attack situations. Examples of non-attack situations here include: a system executing an ordinary app, such as playing a YouTube video, or a high definition movie stored in the local hard disk, or browsing certain websites, copying, or moving a file, or even idling etc. We also noticed that, mouse movements, keyboard entries, clicking to start a program, opening of more YouTube video windows, will also result in high parameter value increases. As to which level the parameter value increases reach depends on the underlying operating systems. We found out that, the windows system is the least affected by these movements, followed by the Linux system, and the OS X is the most affected. And the levels they reached or jumped to in OS X can be even higher than the levels introduced by Spectre attacks.

Another observation that is extremely useful and important is that the value jumps introduced can only be sustained for a very short period of time, usually a few seconds. And this observation forms another basis for our detection mechanism. According to the Spectre information in [1], their non-optimized code can read approximately 10KB/s on an i7 Surface Pro 3 and as we mentioned above, the Meltdown test dumps the kernel memory at up to 503 KB/s. From these facts we know that, to mount a successful attack, an attacker will need quite a protracted duration to get the secrets from a victim.

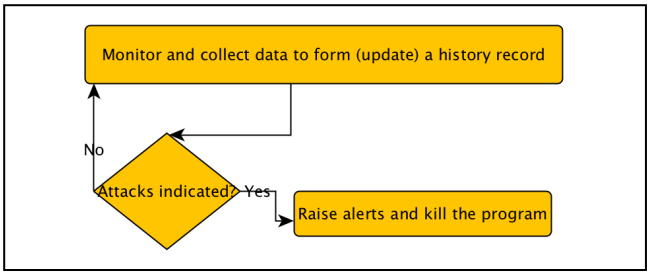

Figure 3. The detection mechanism of Spectre and Meltdown Attacks

Based on the two observations mentioned above, we have formed the defense mechanism with two components: monitoring and detecting. The flowchart in Figure 3 illustrates our defense mechanism. More details of these two components are presented as follows:

1) Monitoring. We monitor the system and measure the parameters and log a historical record with these parameter measurements. The length of this record depends on the effects introduced by the targeted executing app or events (explained above).

2) Detecting. Empirically, we can establish near-optimal thresholds to detect the attacks as soon as possible. Upon continuously exceeding these thresholds, the CELLS detector raises the alert of an ongoing attack. If, on the other hand, the parameters are within the safe ranges (below the set thresholds), our program will update the historical record (adding new data and deleting the oldest data) and continue monitoring the system.

Next, we are going to present the testing results of our defense scheme on different operating systems: Windows, Linux, and OS X.

B. *The Detection on Windows System*

The PoC Spectre and Meltdown attack durations are very short. As we mentioned above, the real attacks will take tens of minutes if not hours in order to get secrets such as passwords, personal IDs, emails, etc., from the victim. We emulate the real attacks by repeating the PoC attacks in loops.

The Meltdown PoC attack we used for windows system is from [18] and the Spectre PoC attack is from [1].

We can see from Figure 4, that the Meltdown attack, can successfully get the value 42, which is private data. And the

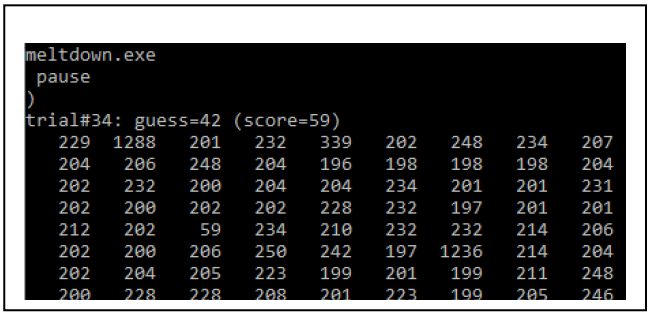

Figure 4. The Meltdown Attack on Windows System

score for it is 59, which is the smallest CPU cycles (cache side channel timing attack) among other numbers in the group. As

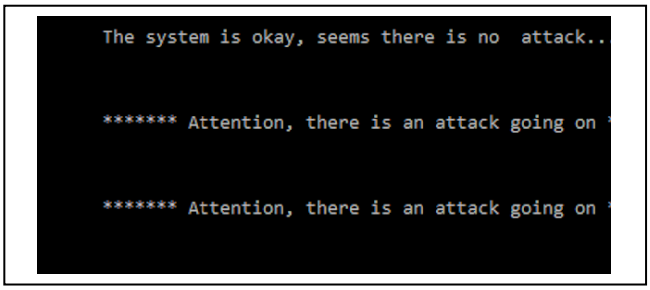

Figure 5. Detection of Attacks on Windows System

we see from Figure 5, that our detector can detect the Meltdown attack, the detector raises alerts within a few

seconds (6 seconds, specifically) to detect the attack. For the Spector PoC attack, we can detect the attack within 5 seconds. It is worth noting here that we use the same detector (i.e., same application with same historical record length and the same threshold) for both Meltdown and Spectre.

C. *The Detection on Linux System*

We use the Meltdown PoC attack from [19] for Linux systems, a similar process that we make it loop to emulate the real attack that our mechanism can detect (Figure 6)

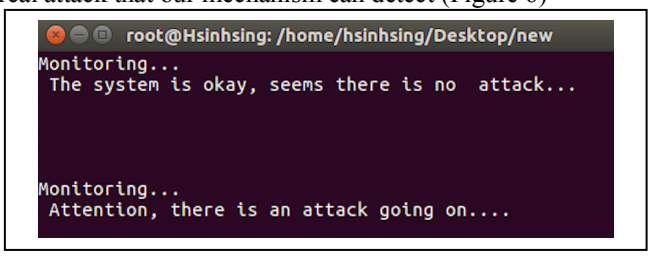

Figure 6. Detection of Attacks PoC on Linux System

the Meltdown attack (Figure 7) in 7 seconds, and 5 seconds for the detection against the Spectre PoC attack.

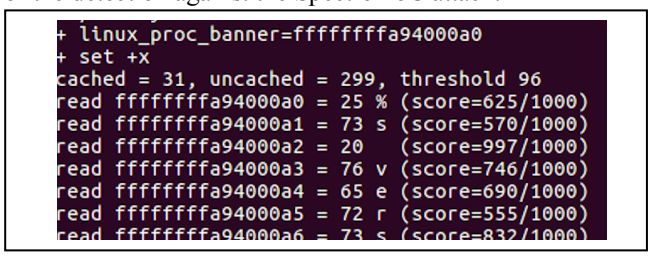

Figure 7. The Meltdown Attack on Linux System

D. *The Detection on OS X System*

As we mentioned about those movements, such as

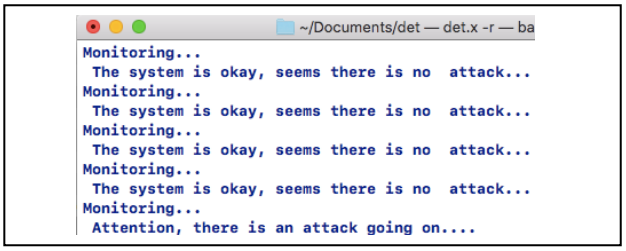

Figure 8. Detection of Spectre Attacks on OS X System

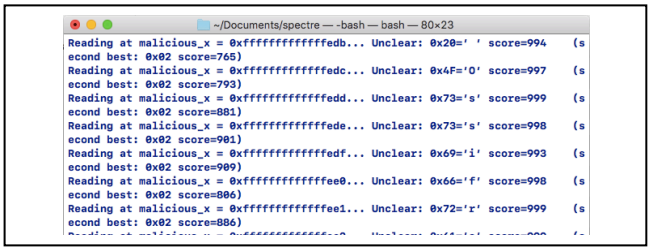

Figure 9. The Spectre Attack [1] PoC on OS X System

moving the mouse, opening a new window and even video watching affects those of statistical parameters and the level of which jumps a lot, especially for OS X systems, so we tried to compensate for these effects by using a longer length for the historical record, and adjusting the threshold accordingly (in

order to minimize the false alarms). However, the side effect is that our detection mechanism needed more time to detect the attack. It takes about 12 seconds to detect the Spectre PoC in Figure 8 and 9.

We don't have a functional Meltdown PoC on OS X system, because all of our OS X systems are already patched. Nevertheless, we believe that our defense mechanism can also detect any real Meltdown attacks, as we have successfully done it on both Windows and Linux Systems (including the demonstrated Spectre attack and its detection on a patched Mac OS X system).

IV. Future Work

In this paper we have demonstrated how Spectre and Meltdown attacks can be detected in real time and across all the major operating systems, by light-weight multi-agent statistical performance monitoring in modern processors architectures. As one part of our future work, we are going to identify which active (i.e., currently running) program(s) or process(s) cause(s) these attacks and how to terminate them before data leak through cache memory attacks. We are also continuing our research in the area to make our defense mechanism more effective by systematic exploration of attack vectors and countermeasure techniques to other Advanced Persistent Threat attacks.

V. Conclusion

The complexity of computing systems has increased rapidly in recent years. Proportionally, the performance of the systems has been increased immensely, as the new software programs adjust to the hardware features such as hierarchical cache subsystems, pipelining, out-of-order execution, speculative execution, etc. However, cleverly crafted exploitations take advantage of the hardware's micro-architecture features and create new classes of vulnerabilities and have the potential to expose data from the desktop to Cloud servers.

The CELLS detection method discussed here is based on statistical performance monitoring of CPU metrics in Intel processors and work in cross-platform settings. By sharing this information, we hope to further engage the research community and identify multilayered options for detection and mitigation against Meltdown, Spectre and other similar side channel attacks. With our promising initial results, we continue to evolve the detection mechanism to handle more advanced attacks.

References

[1] "Meltdown and Spectre". https://meltdownattack.com. Accessed 28 June 2018.

[2] 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.

[3] Peter K.K. Loh, Brian W. Y. Loh, "Cells — A novel IOT security approach," in *Conf. Rec. 2016 IEEE Int. Conf. TENCON,* pp. 3716–3719.

[4] Chandra S. Veerappan, Peter K.K. Loh, Zhao Hui. Tang, Forest. Tan "Taxonomy on Malware Evasion Countermeasures" in *Conf. Rec. 2018 IEEE Int. Conf. WF-IoT,* pp. 558-563.

[5] Vulnerability of Speculative Processors to Cache Timing Side-Channel Mechanism, https://developer.arm.com/support/arm-security-updates/speculative-processor-vulnerability. Accessed 29, June,2018.

[6] Suzaki, K., Iijima, K., Yagi, T., and Artho, C. "Memory deduplication as a threat to the guest Os." In Proceedings of the 2011 European Workshop on System Security.

[7] Osvik, D. A., Shamir, A., and Tromer, E., "Cache Attacks and Countermeasures: The Case of AES.," *In CT-RSA (2006).*

[8] Percival,C, "Cache missing for fun and profit" *In Proceedings of BSDCan (2005).*

[9] Yarom, Y., and Falkner, K., "Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security Symposium (2014).*

[10] Irazoqui, G., Inci, M.S., Eisenbarth,T.,Andsunar, B, "Wait a minute! A fast, Cross-VM attack on AES," in *RAID* 2014.

[11] M. Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T, *"Cross-Tenant Side-Channel Attacks in PaaS Clouds" In CCS'14*, 2014.

[12] Lipp, M., Gruss, D., Spreitzer, R., Maurice, c., and Mangard, S. Armageddon, "Cache Attacks on Mobile Devices," *In USENIX Security Symposium (2016).*

[13] Gruss, D., Maurice, C., Fogh, A., Lipp, M., and Mangard, S, "Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR,"in *CCS (2016).*

[14] Hund, R., Willems, C., Andholz,T., "Practical Timing Side Channel Attacks against Kernel Space," In *S&P.* 2013.

[15] Jang, Y., Lee, S., and Kim, T, "Breaking Kernel Address Space Layout Randomization with Intel," In *CCS (2016).*

[16] CVE-2017-5753, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5753. Accessed 29 June 2018.

[17] CVE-2017-5715, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5715. Accessed 29 June 2018.

[18] Gras, B., Razavi, K., Bosman, E., Bos, H., and Giuffrida, C., "ASLR on the Line: Practical Cache Attacks on the MMU," In *NDSS (2017).*

[19] Msmania, "Meltdown/Spectre PoC for Windows," https://github.com/msmania/microarchitectural-attack. Accessed 28 June, 2018.

[20] Paboldin, "meltdown-exploit," https://github.com/paboldin/meltdown-exploit. Accessed 28 June, 2018.

[21] Sebastian Moss, "Intel admits Meltdown/Spectre patches cause severe reboots," https://www.datacenterdynamics.com/news/intel-admits-meltdownspectre-patches-cause-server-reboots. Accessed 28 June, 2018.

[22] Stephen Nellis, "Intel: Problem in patches for Spectre, Meltdown extends to newer chips," https://www.reuters.com/article/us-cyber-intel/intel-problem-in-patches-for-spectre-meltdown-extends-to-newer-chips-idUSKBN1F7087. Accessed 28 June, 2018.