Exploration for Software Mitigation to Spectre Attacks of Poisoning Indirect Branches

Baozi Chen, Qingbo Wu, Yusong Tan, Liu Yang and Peng Zou

College of Computer Science, National University of Defense Technology, 410073 Changsha, People's Republic of China

ABSTRACT
Speculative execution and branch prediction are techniques that are widely used in modern superscalar processors to exploit instruction-level parallelism. Recently, researchers have discovered a new kind of attacks named Spectre which exploits speculation mechanisms with a side channel. Since speculation is widely used in modern superscalar processors, these vulnerabilities are found in many popular processors. Exploiting the security vulnerabilities, the attacker can leak the host memory from inside a KVM guest. While the hardware providers are trying to fix the issues from the microarchitecture designs in the next generation of their products, software mitigation are always desirable. Retpoline is a pure software fix developed by Google and is claimed to have a negligible impact on performance. In this paper, we look into the details of Retpoline and evaluate it with various workloads. We found that Retpoline does have impact on performance to the existing software but varies depending on how applications interact with the kernel. According to our experiment, it shows more regression on the network I/O than the storage. The more a program relies on the kernel, the greater regression it shows. To alleviate the impact, we propose a method that uses userspace network stack. We verify the proposal using Netmap userspace packet I/O framework. Besides, we observe great performance regression of applying Retpoline in some of userspace applications such as Perl interpreter which is thought to be the targets exploited by the new type of attacks. In the end, we review the experiment results and discuss the potential mitigation of Spectre in future.

KEYWORDS
Indirect branch prediction; Out-of-order execution; Retpoline; Security vulnerability; Spectre; System performance

1. INTRODUCTION
Modern superscalar processors are designed to exploit instruction-level parallelism (ILP). Speculative execution and branch prediction are the two most widely used techniques to maximize performance. Speculative execution allows the processor to execute instructions before being certain whether this execution should happen, while branch prediction can predict the next instruction to fetch and avoid stalling of control dependencies.

Traditional security researches try to avoid sensitive information leakage through covert channels by restricting information-flow policies [1] or data encryption [2]. However, attackers can bypass those restrictions or encryptions and leak sensitive data through the side channel. Since personal multimedia information is stored in the cloud service, privacy protection becomes increasingly important [3–5]. Recently, researchers have discovered a new kind of attacks named Spectre which exploits speculative behavior of modern processors and side channels [6, 7]. By training the branch predictor, the adversary can misdirect the processor to execute unauthorized code speculatively and leak information through side channels. There are mainly two variants of Spectre attacks. One leverages conditional branches, and the other leverages indirect branches. Mitigating new attacks is thought to be expensive at the moment because it is based on common optimization techniques on modern processors. Since Spectre utilizes speculative execution and cache, simply disabling either of two would cause great degradation of performance.

Engineers from Google propose a pure software fix called Retpoline [8, 9], which addresses the Variant 2 of Spectre and claims that it generally has negligible impact on performance. Retpoline replaces the vulnerable indirect branch instruction with a trampoline. It utilizes the function return instruction which does not depend on the branch target buffer (BTB) to implement an indirect branch. In this paper, we evaluate Retpoline with various benchmarks for both kernel and userspace applications. We found that Retpoline has impact on the performance of the existing software but varies depending on how applications interact with the kernel. According to our experiment, it shows about 7% regression on the network I/O, while there is little overhead observed on
storage I/O. Since the performance regression of Retpoline is brought in through invoking kernel in applications, SPECCPU benchmark is not affected. The more the applications rely on the kernel, the greater the regression. To alleviate the impact, we propose a method that uses userspace network stack. We evaluate the proposal using Netmap [10] userspace packet I/O framework. By using Netmap userspace network stack, the TCP transmission rate can achieve almost the same throughput compared to the non-Retpoline system. In the end, we review the experiment results and discuss the potential mitigation of Spectre in future.

The rest of the paper is organized as follows. First, we describe the background of the new attacks and existing mitigations in Section 2. Then we introduce Retpoline techniques and explain how it can be used to mitigate the Variant 2 of Spectre in Section 3. Later, we illustrate the setup of our evaluation experiment and results of our tests in Section 4. In the end, we analyze the results and make our discussion in Section 5.

2. BACKGROUND

This section introduces the background of the new attack methods. We first describe the micro-architecture mechanism behind it and the concept of the side channel. Then we look into the details about Spectre and briefly review existing mitigations.

2.1 Speculation with Branch Prediction

Exploiting ILP is one of the key goals in high performance processor design. Many techniques have been developed to maximize ILP. Branch prediction and speculative execution are two representative methods and have been adopted widely in modern superscalar processors.

When a branch instruction is executed, it may change the original increment of PC which is used immediately in the fetch stage of the next instruction in pipeline. Since the new value of PC would be determined only after the branch instruction has been decoded, there would be stalls for the next instruction to wait for the PC value of the next fetch. Branch prediction is done by the processor to try to determine where the execution will continue after a conditional jump, so that it can read the next instruction from memory without any stalls. Speculative execution goes one step further and determines what the result would be from executing the next instructions. If the branch prediction was correct, the result is used, otherwise it is discarded. A common technique to improve the efficiency of speculation is to only speculate on instructions from the most likely execution path, which means to combine with branch prediction.

In other words, branch prediction determines which execution path to choose. Speculation executes instructions along the path while the condition that depends on the result of previous instructions is still being determined in the pipeline.

2.2 Side-Channel Attacks

Side-channel information is the information that can be obtained from the physical states of a system rather than the plaintext of information itself. A side-channel attack targets to utilize the side-channel information to perform attacks rather than the weakness in the system implementation itself. Depending on the physical characteristics of systems, there are various kinds of side channels which can be used by the adversaries to leak sensitive information. Timing information [11], power consumption [12, 13] and electromagnetic leaks [14] are all sources of information and can be exploited to perform side-channel attacks.

Modern processors use cache to fill up the speed gap in memory hierarchy. At the same time, it introduces uncertainty to the system that time of memory accesses varies depending on whether the data are in cache or not. Cache timing attacks are a specific type of side-channel attack that exploit the effects of the cache memory on the execution time of algorithms. The attacker can determine the addresses allocated into cache by measuring the time taken to access entries and leak information. There are several techniques to exploit cache that have been demonstrated already. PRIME+PROBE [15–17] is the one that the attacker fills one or more cache lines with its own contents, waits for the victim to execute and then probes by timing accesses to preloaded cache lines. If the attacker observes remarkable increased memory access latency, it means that the cache lines have been evicted by the victim who has touched an address that maps to the same set. FLUSH+RELOAD [18] is contrast to PRIME+PROBE. The attacker first flushes targeted cache lines, waits for the victim to execute and then reloads the flushed cache line by touching it, in the meanwhile measuring the time taken. If the attacker observes a fast memory access, it means that the cache lines have been reloaded by the victim. EVICT+TIME [19] compares the overall execution time of the victim after evicting some cache lines of interest with a baseline. The variation of overall execution time is then used to deduce whether the lines of interest have been accessed by the victim.
2.3 Spectre Attacks

From a security point of view, speculative execution with a wrong prediction path of branch may lead the processor to execute some malicious codes that should not have been done. Although the roll-back mechanism of speculation guarantees that all the states in the processor cores modified by speculative execution would be restored when the prediction fails, the side channels such as caches may still retain some modifications which might be used to leak the sensitive information unintentionally.

By performing speculative memory accesses to cacheable locations beyond an unresolved branch in Spectre attacks, the result of those reads can themselves be used to form the addresses of further speculative memory reads. The effects of the second speculative allocation within the caches can then be measured by the adversaries and leads to leakage in sensitive information. According to the research, Spectre attacks can be exploited using both native codes and JavaScript.

There are two variants of Spectre attacks. Variant 1 exploits conditional branches speculation and can be used to bypass software sanitization of offsets. Figure 1 shows a code example of variant 1. During the speculative execution, a processor may mispredict the direction of the conditional branch and make the processor speculatively execute the incorrect path of codes. After the speculative execution, the cache line holding arr2->data[index2] remains in the cache. The attackers can then determine whether the value of index2 was 0x200 or 0x300 by timing the loading of arr2->data[0x200] and arr2->data[0x300], which can be used to deduce whether arr1->data[untrusted_offset]&1 is 0 or 1.

Figure 1: Conditional branch code example [20]

2.4 Existing Mitigations

Since the Spectre is rooted in the hardware itself, the final solution should count on the update designs of the processors.

Mitigating conditional branch (variant 1) requires that speculation is halted on potentially sensitive execution paths, which usually exist in JIT engine. A static analysis tool could be helpful to excavate potential vulnerable pattern among the codes. After the pattern is located, a barrier can be inserted to stop speculation. For x86 system, the LFENCE instruction is desired to perform serializing which stop younger instructions from executing, even speculatively, before older instructions have retired [22, 23]. AMD also proposed a method that creates a data dependency to avoid speculative execution [23]. Dong et al. [24] proposed an instrumentation-based bit masking approach with Intel MPX mechanism to defense against in-kernel speculation side channel. ARM suggests that inserting the combination instructions of an DSB SYS and an ISB to prevent speculation in the old system and proposes a new instruction CSDB which has better performance in the next-generation architecture [25].

Indirect branch poisoning (variant 2) is considered to be more challenging to mitigate. It is widely believed that new hardware mechanisms are required to fix the issue at a minimum cost. Intel suggests a new interface between the processor and system software that allows system software to prevent an attacker from controlling the victim’s indirect branch prediction which requires both updated system software and microcode [22]. AMD defined additional steps through a combination of processor microcode updates and operating system patches to mitigate the threat [23]. In addition, it also introduces Indirect Branch Prediction Barrier (IBPB), Indirect
Branch Restricted Speculation (IBRS) and Single Thread Indirect Branch Prediction mode (STIBP) for the next generation of processors to fix the issues [26]. Although there are plenty of implementations for ARM architecture, which means only part of them might be affected and makes no generic mitigation that applies for all ARM processors, many ARM processors have implementation-specific controls that can be used either to disable branch prediction or to provide mechanisms to invalidate the branch predictor [25]. Besides hardware-side mitigation, there is also a pure software mitigation of variant 2 called Retpoline [8, 9] developed by Google which is said to have an observed negligible impact on performance. In the following sections, we will analyze it in detail.

3. RETPOLINE TECHNIQUES

Retpoline is a portmanteau of “return” and “trampoline”. In this section, we first describe the concept of trampoline technique. And then we analyze the implementation details of Retpoline and how it protects the Linux operating system from variant 2.

3.1 Trampoline

A trampoline, also known as thunk [27], refers to a small piece of code called as a function, does some small work, and then jumps to another location (Figure 2). Unlike ordinary function call, it does not return to its caller after execution, as shown in Figure 3. If the jump target of the trampoline is a function which means it will jump back to some location, it will return to the trampoline’s caller. It can be used as calling convention translation, virtual function translation or dynamic closures.

3.2 Retpoline: An Indirect Branch Trampoline

With speculation, the factors that may determine the following execution sequence of an indirect branch depend on two parts. One is the final branch target address, the other one is the result of branch predictors. Among all indirect branch instructions, function return instruction has a unique character. Recall that function return is itself an indirect branch. Unlike ordinary indirect branches, the speculation path of function return is determined not by itself but by the paired function call. It is generally assumed that a function should return to the address next to the function call instruction. Thus, the prediction target of function return is determined by the function call and are recorded in return stack buffer (RSB) separately [28], which means that BTB collision poisoning cannot affect it. However, the actual destination of function return can be later modified through the stack or link register.

Retpoline sequences are a software construct that allows indirect branches to be isolated from speculative execution. It utilizes the property of function return and provides a controlled speculative execution for indirect branches. It uses properties of the RSB, which is filled with safe targets on entry to a privileged mode, to control speculation. RSB is located in dispatch stage and stores return address when CALL instruction is dispatched and provides return address when RET instruction is dispatched. Contents of RSB are filled once the CALL instruction is dispatched. As shown in Figure 3, Retpoline first invokes a CALL to set_up_target which sets the prediction target of RET to capture_spec. Then it modifies the return target address to the content of r11 on the stack and dispatch RET. Before the final branch is

```plaintext
CALL set_up_target ;(1) "capture_spec" is saved to RSB
capture_spec: ;(4) speculative execution of RET
PAUSE
JUMP capture_spec
set_up_target:
MOV tr11, (%esp) ;(2) modify the return target on the stack
RET ;(3) jump to the target
```

Figure 2: Trampoline refers to a piece of code that would jump to another location after finishing execution rather return to the original caller.

Figure 3: Indirect branch construction for “JMP *%r11” [8]
Table 1: UnixBench benchmark tests description

| Tests               | Descriptions                                                                 |
|---------------------|-----------------------------------------------------------------------------|
| Dhrystone           | A synthetic computing benchmark program used to measure the integer operations, which consists of a set of string handling in scientific applications |
| Whetstone           | A test contains several modules that are meant to represent a mix of operations typically performed |
| exeql Throughput    | Measure the number of exeql calls that can be performed per second           |
| File Copy           | Measures the rate at which data can be transferred from one file to another, using various buffer sizes |
| Pipe Throughput     | The number of times per second a process can write to a pipe read them back |
| Pipe-based Context Switching | The number of times two processes can exchange an increasing integer through a pipe |
| Process Creation    | Measure the number of times a process can fork and reap a child that immediately exits |
| Shell Scripts       | Measure the number of times per minute a process can start and reap a set of eight concurrent copies of a shell scripts where the shell script applies a series of transformation to a data file |
| System Call Overhead| Estimates the cost of entering and leaving the operating system kernel       |

done by RET, the processor would speculatively execute the loop pointed by capture_spec.

3.3 Protecting Linux with Retpoline

Applying Retpoline to protect from Spectre Variant 2 means using Retpoline construction to replace indirect branch instructions. Converting all indirect branches in the kernel requires compiler support [29, 30] which inserts Retpoline instead of indirect branch instructions in the backend. Besides, there are some fragments in the kernel implemented in the assembly. These include system call entries, IRQ inline assembly, hyper calls and so on, which consist of a basic boundary for context switch. Without compiler support, it can still make up a minimum protection against Variant 2 attacks in the kernel.

4. EVALUATION

4.1 Methodology

To further understand Retpoline, we have run several benchmarks to evaluate its impact on performance. We choose a mainstream Intel platform (Core i7-3770@3.40 GHz) as our target hardware. We fetch and build the latest v4.15 Linux kernel with GCC 7.3.0 from Git repository with and without the Retpoline configuration. Under these two kernels, we launch various tests targeting performance from I/O to userspace application.

We first run UnixBench to get a generic view of system performance. UnixBench provides a basic indicator of the performance of a Unix-like operating system and is widely used to evaluate system performance. Table 1 shows the details of UnixBench test suit. For I/O performance, dbench is chosen to evaluate storage workloads while tbench is used to measure network stack. Dbench is an emulation of the file system load function of the Netbench benchmark. It does all the same I/O calls that a Samba server would produce. While dbench produces only the file system load, it does no networking calls. In order to simulate networking loads, tbench is developed along with dbench. It produces only the TCP and process load which does the same socket calls that the Samba server would do under a netbench load. This benchmark is a good simulation of a real server setup, in which a large amount of files with different sizes and directions have to be created, written, read and deleted. Both tests are run for ten times and then are taken the average for comparison. Finally, we run SPECCPU2006 under those two kernels to survey the impact of in-kernel Retpoline to pure computing in userspace. The SPECCPU2006 benchmark is an industry-standard benchmark that is designed to compare the performance of servers. The workloads do not require extremely large amounts of memory, and since there is essentially no disk or network I/O. It measures performance on two different basic workloads: one based on compute-intensive integer math (Table 2), and the other based on compute-intensive floating-point math (Table 3). In general, the integer workload maps to the performance of business applications found in the data center, and the floating-point workload maps to scientific calculations more often found in a high-performance computing environment.

In addition, we rebuild SPECCPU2006 with Retpoline support in userspace to evaluate performance impact, since Spectre may be exploited to attack not
Table 3: SPECfp 2006 benchmark descriptions

| Tests          | Descriptions            |
|----------------|-------------------------|
| 410,bwaves     | Fluid dynamics          |
| 416,gamess     | Quantum chemistry       |
| 433,Milc       | Quantum chromodynamics  |
| 434,zeusmp     | Physics                 |
| 435.gromacs    | Fluid dynamics          |
| 436.cactusADM  | Physics – general relativity |
| 444.namd       | Fluid dynamics          |
| 447.dealII     | Biology – molecular dynamics |
| 450.soplex     | Finite-element analysis |
| 453.povray     | Linear programming      |
| 454.calculix   | Ray-tracing             |
| 459.GemsFDTD   | Structural mechanics    |
| 465.tonto      | Computational electro-magnetics |
| 470.lbm        | Quantum dynamics        |
| 481.wrf        | Fluid dynamics          |
| 482.sphinx3    | Weather prediction      |
| 486.tonto      | Speech recognition      |

Figure 4: Normalized results of UnixBench for Linux Kernel v4.15.0 with Retpoline patches

Figure 5: Normalized results of I/O pesyformance for Linux Kernel v4.15.0 with Retpoline patches

Figure 6: Overview of SPECCPU2006 results for Retpoline

4.2 Results

For comparison, we normalized all the results. Figure 4 shows the result of UnixBench which provides a basic indicator of the performance of a Unix-like system in various aspects. Not all the micro benchmarks are affected by the Retpoline kernel. Tests such as Dhrystone2 and Whetstone that targets at pure computing show almost the same under either of the kernel. Those tests such as file copy, pipe operation and system call that depend on the kernel function suffer from Retpoline kernel in various degrees.

As shown in Figure 5 Retpoline kernel influences I/O subsystem of kernel. While there is little overhead on storage, regression on network is more remarkable. TCP transmission rate on Retpoline kernel is about 93% compared to the one without the patches.

We have run SPECCPU2006 for three times under different setups. Firstly, it is run on the 4.15.0 Linux kernel without any Retpoline options, of which the result would be used as the baseline. Secondly, we launch the benchmark on the 4.15.0 Linux kernel with Retpoline enabled (aka in-kernel Retpoline). In the end, we rebuild the SPECCPU2006 test set with Retpoline compiler options for userspace application and benchmark under the Retpoline-enabled kernel. As we can see from Figure 6, in-kernel Retpoline has little impacts on the results of SPECCPU which indicates that compute-intensive application may be run unaffected with the mitigation. However, applying Retpoline in userspace which may be used to reinforce Spectre-affected applications could suffer from performance reduction in certain areas. Since there is little regression on floating point applications, it would greatly impact certain integer tests. Figures 7 and 8 show the details of SPECCPU2006 testing items. For SPECint2006, 400.perlbench is affected most which shows less than 60% performance when applying Retpoline in userspace. Considering that interpreter along with JIT engine is one of the targeted software, it is worth to carefully adopt Retpoline as the mitigation.
4.3 Userspace Network I/O

Previous report [31] shows that networking-related CPU overheads for a kernel-based TCP stack can be up to 40% for application context switching. According to the benchmark results, there is a significant impact on network I/O but negligible regression on filesystem and storage due to the protocol processing in kernel which use indirect branches more than storage system. Userspace network stack has been proposed to improve the cache performance on multi-core systems [32], meanwhile avoiding extra data copies and boundary crossings. Since we can apply Retpoline to kernel and leave userspace network driver speculatively executing indirect branch as usual, performance can benefit from userspace network stack. Various frameworks such as Netmap provide efficient packet reception and transmission mechanisms to or from user space bypassing kernel stack packet processing. These frameworks reduce or remove various packet processing costs such as per-packet dynamic memory allocations, system call overheads and memory copies to userspace.

Processed by the kernel, a packet arrives at the network card's interrupt handler. As soon as the driver acknowledges the interrupt controller, the remainder of the work may well be performed in a tasklet. Later, the core network processing happens in another software interrupt which would consist of different stages in multiple layers. In the end, the data are copied to the userspace resulting in context switch. A whole network transaction would cover a series of packet exchanges through multiple system calls which includes not only I/O functions but also security audits. This process would pass through much more indirect branches than others. Thus, it suffers more regression from Retpoline which prevents speculation of indirect branches. In this case, we propose to use a userspace network stack to alleviate such regression. Netmap maps NIC buffers directly into userspace memory and avoids the memory copy between the kernel and userspace applications. Thus, the application can access a packet as simple as reading from the buffer in the mapped region. Once the buffers are ready for the NIC, the application can then initiate DMA operations of the NIC through `ioctl`. To certify our proposal, we measure the throughput of Netmap under the two kernels. Figure 9 shows the network throughput when applied Netmap network stack. Netmap within-kernel Retpoline achieves almost the same throughput compared to the non-Retpoline system.

5. DISCUSSIONS

As we saw in the previous section, Retpoline does not bring about severe regression to the whole system in general. However, it does impact performance in certain circumstances, especially when applied to userspace program. Since Spectre does not only affect the kernel, userspace program such as JIT engine may also require for Retpoline under certain conditions.

To carry out attacks against Spectre, two conditions should be met, which are speculative execution and side channel. Retpoline belongs to a kind of techniques that prevent speculative execution. As we have seen in this paper, fully protection by Retpoline also causes some regression. To cut down regression, one way is to restrict the scope of the impact, for example, limiting uses of Retpoline only in kernel, or even only in some key path of
context switching. In the same vein, cutting off the side channel may have similar effects. Since Spectre relies on the side channel of caches which plays an important role in the performance on modern CPU, it would be unacceptable to shut down this side channel to prevent attacks. However, if we can identify that only a small portion of code may suffer from the vulnerability and it is not in the critical path, shutting down the side channel can also be an option.

6. CONCLUSION

Exploiting speculative execution to perform a side-channel attacks arises a new type of threat which can be used to leak sensitive information unintentionally from a privileged domain regarded as safe previously. Since speculation is widely used in modern superscalar processors, these vulnerabilities are found in many popular processors including Intel, AMD and some implementations of ARM. While the hardware providers are trying to fix the issues from the microarchitecture designs, software mitigations are always desirable since it is much cheaper and can be applied in legacy system. In this paper, we evaluate one of the software mitigations called Retpoline that can be used to protect from attacks which exploit indirect branch speculation. We found that in-kernel Retpoline has an impact on performance to the existing software but varies depending on how applications interact with the kernel. To alleviate the impact, we propose a method that uses userspace network stack. We evaluate it using Netmap userspace packet I/O framework and verify the proposal. Besides, we observe great performance regression of applying Retpoline in some of userspace applications such as Perl interpreter which is thought to be the targets exploited by the new type of attacks.

FUNDING

This work was supported by the National Science & Technology Major Project of China [2016ZX01040101]; National Natural Science Foundation of China [61602498 & 61872444].

ORCID

Baozi Chen http://orcid.org/0000-0001-8608-1603

REFERENCES

1. J. Cui, Y. Zhang, Z. Cai, A. Liu and Y. Li, “Securing display path for security-sensitive applications on mobile devices,” CMC: Comput. Mater. Continua, Vol. 55, no. 1, pp. 17–35, 2018.

2. W. Xu, S. Xiang, and V. Sachney, “A cryptograph domain image retrieval method based on Paillier Homomorphic block encryption,” CMC: Comput. Mater. Continua, Vol. 55, no. 2, pp. 11–21, 2018.

3. Y. Ma, X. Luo, X. Li, Z. Bao, and Y. Zhang, “Selection of rich model steganalysis features based on decision rough set - positive region reduction,” IEEE Trans. Circuits Syst. Video Technol., pp. 1–1, 2018. doi:10.1109/TCSVT.2018.2799243.

4. J. Wang, T. Li, Y. Shi, S. Lian, and J. Ye, “Forensics feature analysis in quaternion wavelet domain for distinguishing photographic images and computer graphics,” Multimed. Tools Appl., Vol. 76, no. 22, pp. 23721–37, 2017.

5. Y. Zhang, C. Qin, W. Zhang, F. Liu, and X. Luo, “On the fault-tolerant performance for a class of robust image steganography,” Signal Process., Vol. 146, pp. 99–111, 2018.

6. P. Kocher, et al., “Spectre attacks – Exploiting speculative execution,” meltdownattack.com, 2018.

7. T. M. Conte, E. P. DeBenedictis, A. Mendelson, and D. S. Milojicic, “Rebooting computers to avoid meltdown and spectre,” IEEE Comput., Vol. 51, no. 4, pp. 74–7, 2018.

8. P. Turner, Retpoline: A software construct for preventing branch-target-injection. Available: https://support.google.com/faqs/answer/7625886.

9. Woodhouse. D., Linux Retpoline. Available: http://git.infra debian.org/users/dwmw2/linux-retpoline.git.

10. L. Rizzo, “netmap – A novel framework for fast packet I/O,” in USENIX Annual Technical Conference, 2012, pp. 9–9.

11. P. C. Kocher, “Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems,” in Advances in Cryptology — CRYPTO ’96. Berlin, Heidelberg: Springer, Jul. 2001, pp. 104–13.

12. P. Kocher, J. Jaffe, and B. Jun, “Introduction to differential power analysis,” J. Cryptogr. Eng., Vol. 1, no. 1, pp. 5–27, Mar. 2011.

13. P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in Advances in Cryptology — CRYPTO ’96. Berlin, Heidelberg: Springer, Dec. 1999, pp. 388–97.

14. J.-J. Quisquater and D. Samyde, “ElectroMagnetic Analysis (EMA): Measures and counter-measures for smart cards,” in Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security. London, UK: Springer-Verlag, 2001, pp. 200–10.

15. C. Percival, “Cache missing for fun and profit,” in BSDCan, 2005.

16. D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures – The case of AES.” CT-RSA, 2006.

17. F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, “Last-level cache side-channel attacks are practical,” in 2015 IEEE
Symposium on Security and Privacy (SP). IEEE, Apr. 2015, pp. 605–22.

18. Y. Yarom and K. Falkner, “‘FLUSH+RELOAD – A high resolution, low noise, L3 cache side-channel attack,’” in USENIX Security Symposium, Vol. 1, 2014, pp. 22–5.

19. M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, “‘ARMageddon – Cache attacks on mobile devices,’” in USENIX Security Symposium, 2016, pp. 549–64.

20. J. Horn, Reading privileged memory with a side-channel, Jan. 2018. Available: https://goo.gl/TKBGi9.

21. D.Evtushkin, D. Ponomarev, and Abu-Ghazaleh N., “Jump over ASLR: Attacking branch predictors to bypass ASLR,” in 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, pp. 1–13.

22. Intel, “Intel analysis of speculative execution side-channels,” Technical Report 336983-001, Jan. 2018.

23. AMD, “Software techniques for managing speculation on AMD processors,” Technical Report, Feb. 2018.

24. X. Dong, Z. Shen, J. Criswell, A. Cox, and S. Dwarkadas, “Spectres, virtual ghosts, and hardware support,” in Hardware and Architectural Support for Security and Privacy (HASP) 2018. New York, NY, USA: ACM Press, 2018, pp. 1–9.

25. ARM, “Cache speculation side-channels,” Technical Report, Feb. 2018.

26. AMD, “Indirect branch control extension,” Technical Report, Feb. 2018.

27. P. Z. Ingerman, “Thunks – A way of compiling procedure statements with some comments on procedure declarations,” Commun. ACM, Vol. 4, no. 1, pp. 55–8, 1961.

28. Intel Corporation, Intel64 and IA-32 architectures optimization reference manual, Jun. 2016.

29. H. J. Lu, x86: CVE-2017-5715, aka Spectre. Available: https://gcc.gnu.org/ml/gcc-patches/2018-01/msg00422.html.

30. C. Carruth, Introduce the “retpoline” x86 mitigation technique for variant #2 of the speculative execution vulnerabilities. Available: https://reviews.llvm.org/D41723.

31. B. Hausauer, iWARP: Reducing ethernet overhead in data center designs, Nov. 2014. [Online]. Available: http://www.eetimes.com/design/communications-design/4009333/iWARP-Reducing-Ethernet-Overhead-in-Data-Center-Designs.

32. B. F. V. Jacobson, Network channels, Jan. 2006. Available: https://lwn.net/Articles/169961/.

Authors

Baozi Chen received the BS and MS degrees at College of Computer Science from National University of Defense Technology in 2008 and 2012, respectively. Now he is a PhD candidate at National University of Defense Technology. His primary research interests include operating system, virtualization and system security.

Corresponding author. Email: chenbaozi10@nudt.edu.cn

Qingbo Wu received PhD in computer science and technology from National University of Defense Technology in 2010. Now he is a professor and master supervisor at National University of Defense Technology. His research interests include operating system and cloud computing.

E-mail: wqb123@nudt.edu.cn

Yusong Tan received PhD in computer science and technology from National University of Defense Technology in 2004. Now he is a professor and master supervisor at National University of Defense Technology. His research interests include cloud computing and big data.

E-mail: ystan@nudt.edu.cn

Liu Yang received PhD in electronic science and technology from National University of Defense Technology in 2015. Now she is an assistant professor at National University of Defense Technology. Her research interests include system software and compiling optimizing.

E-mail: yangliujoy@nudt.edu.cn

Peng Zou received PhD from National University of Defense Technology. He is a professor and PhD supervisor at National University of Defense Technology. His main research interests include network & information security and distributed computing.

E-mail: zpeng@nudt.edu.cn