CYPRUS INTERNATIONAL UNIVERSITY

DEPARTMENT OF COMPUTER ENGINEERING

INVESTIGATION OF MICROPROCESSOR OR HARDWARE SECURITY FLAWS

By

Jimoh Babatunde OLAWALE

STUDENT NUMBER

21905060

COURSE TITLE: OPERATING SYSTEM AND NETWORK SECURITY

COURSE CODE: CPE 526

COURSE LECTURER

DEVRIM SERAL

Abstract

Microprocessor or hardware security flaws can put your business and your customer sensitive data at risk, costing you in diminished sales, reputation loss and penalties. Present day computer is getting faster, more efficient and progressively interconnected with every age. Consequently, this stage grows more complex, with continuously new features introducing the possibility of new flaws. Most of these flaws arise from continued use of legacy systems and out-of-date software, the fact that the majority of these loopholes don't necessarily raise a red flag may allow hackers to steal information, inject malware or completely hijack your applications or corporate systems. This paper investigates the existing literature in microprocessor or hardware security flaws.

1. INTRODUCTION

All the operations on a computer, at the lowest level, are hardware dependent. Over the course of time computers have evolved from simple computing machines to very complex machines, thanks to the evolution of software and the hardware that drives it. The divide between hardware and software security research is starting to take its toll, as we are witnessing increasingly sophisticated attacks that are combining software and hardware vulnerable to exploit computing platforms at runtime, J.Van Bulck et.al (2018).

Hardware vulnerabilities. Existing security mechanisms are completely circumvented (D. Evtyushkin et al.2018), by cross-layer attacks due to the exclusive focus on mitigating attacks that exploit software vulnerabilities.

The affected targets range from low-end embedded devices to complex servers, that are hardened with architectural defenses such as data-execution prevention, supervisor-mode execution prevention, and advanced defenses such as control flow integrity. Hardware vulnerabilities. Existing security mechanisms are completely circumvented, D. Evtyushkin et al. (2018), by cross-layer attacks due to the exclusive focus on mitigating attacks that exploit software vulnerabilities.

And a significant amount of research has been aimed at analyzing the effects of high energy particles on microprocessor devices. However, less attention has been given to the intermittent faults. Field collected data and failure analysis results clearly show that intermittent faults are a major source of errors in modern integrated circuits. The root cause for these faults ranges from manufacturing residuals to oxide breakdown. Burstiness and high error rates are specific manifestations of the intermittent faults. They may be activated and deactivated by voltage, frequency, and operating temperature variations. T As a result, hardware implemented fault tolerant techniques, such as error detecting and correcting codes, self-checking, and hardware implemented instruction retry, are necessary for mitigating the impact of the intermittent faults, both in the case of microprocessors, and other complex integrated circuits Constantinescu, C. (2008, January).

Moreover, hardware security extensions such as Sanctum, D.Evtyushkin et al.(2018). Their implementation remains vulnerable to potentially undetected hardware bugs committed at design time, and in fact, SGX and TrustZone have been targets of successful cross-layer attacks, J.Van Bulck et.al. (2018). While Sanctum's formal model offers provable security guarantees, its trusted abstract platform model is formulated at a high level of abstraction. As the hardware and software have evolved, so are the vulnerabilities associated with them. It is commonly observed that software vulnerabilities are detected frequently and patched, but same is not the case with hardware vulnerabilities. Vulnerabilities in hardware architecture are not much researched as extensively as software, apart from its developers, in the cyber security domain. This leads to many hardware-based attacks such as

2 HARDWARE-ORIENTED ATTACKS

In design flows commonly used in industry, the digital designer describes the logic behavior of the processor in a clock cycle-accurate way and defines for each instruction the elementary steps of its execution based on the processor's registers. Such design descriptions at the "Register-Transfer Level (RTL)" are often referred to as the microarchitecture of a processor. Numerous degrees of freedom exist for the designer in choosing an appropriate microarchitecture for a specification of the processor given at the level of its instruction set architecture (ISA). However, the same degrees of freedom that the designer uses for optimizing a processor design may also lead to microarchitectural side effects that can be exploited in security attacks.

In fact, it is possible that, depending on the data that is processed, one and the same program may behave slightly differently in terms of what data is stored in which registers and at which time points. These differences only affect detailed timing at the microarchitectural level and have no impact on the correct functioning of the program at the ISA level, as seen by the programmer. However, if these subtle alterations of program execution at the microarchitectural level can be caused by secret data, this may open a "side channel". An attacker may trigger and observe these alterations to infer secret information. In microarchitectural side channel attacks, the possible leakage of secret information is based on some microarchitectural resource which creates an information channel between different software (SW) processes that share this resource. For example, the cache can be such a shared resource, and various attacking schemes have been reported which can deduce critical information from the footprint of an encryption software on the cache Y. Yarom et al. (2017). Also other shared resources can be (mis-)used as channel in side-channel attacks, as has been shown for DRAMs, P. Pessl,(2017) and other shared functional units, O. Aciicmez and J.-P. Seifert, (2007).

This section reviews two well-known attacks which exploit the microarchitecture, specter and meltdown and other microprocessor flaws. In our computers it is the software that has vulnerabilities and tends to believe that hardware is secure, without any vulnerabilities. Just like as software evolves from its older versions adding new features, the processor has also evolved in terms of design and complexity. Over this evolution, vulnerabilities have hidden under the complex architecture only to be discovered and possibly exploited.

2.1 Side channel attack: Prime+Probe

Prime+Probe is a side channel attack against the last-level cache (LLC) which can have cross-core, cross-VM covert channel, Liu F et al. (2015). In this attack, the attacker monitors the cache usage of the victim to determine the cache lines used by the attacker to extract data. The spy or attacker fills the cache set with its own data. This is called as Prime phase. The attacker lets the victim run itself, and it is expected that the victim's code will conflict with the cache lines used by the attacker, it will evict those lines with its own data or code. The attacker then again tries to load its data and monitors the time required to fetch and thus can conclude which lines have been evicted. This is called as Probe phase.

2.2 Side Channel Attack: Flush+Reload

Flush+Reload is a cache-based side-channel attack technique that exploits this weakness to monitor access to memory lines in shared pages, YaromY et al. (2014). This attack utilizes LLC which is furthest from the CPU core. The spy code flushes all the cache lines, then allow the victim to execute and access the memory. If shared memory is accessed, it will be cached. Later the spy code tries to access the same memory while measuring the access time. If access time is considerably low, then it is concluded that a particular memory location was accessed, this information is used to identify the actual data accessed by the victim. Example in which a spy code tries to extract data from the victim's address space. The spy flushes all the shared pages from the cache to mount an initial phase of attack. The victim uses some data from its address space to access content from the shared page. This page when accessed will be cached for future use. The spy can iterate over all the possible values which fetch the shared pages. If there is a cache hit, then spy can conclude that the particular value also exists in the victim's address space.

2.3 Microarchitectural attack: Spectre

Spectre attack causes a victim to speculatively access data which is not accessed in normal execution and leak it over the side-channel, Kocher P et al.(2019). The variants of the spectre are based on how the microarchitecture for speculative execution is exploited. Variant 1 exploits the conditional branches whereas variant 2 exploits indirect branches. The spectre attack can be divided into three important phases, training the branch for speculative execution, accessing data through invalid memory reference, and finally mounting a side-channel attack to extract the data. In variant 1, jump address for the conditional branch can be trained and the processor can be made to execute instructions even if the branch direction is invalid. Listing .1 is from POC Anonymous, (2019). for spectre, in which variable x is attacker controlled, variable x is immediately available whereas array size is not cached and is required to be fetched from memory. In the first phase, by executing the branch multiple times, the jump address is stored in BTB which leads to speculative execution of "if" block. The attacker needs to have shared mechanism to extract data from side channel and thus web browsers are a viable target but this can be mitigated by site isolation, Reis C (2019). The impact of spectre attack can be explained by the following example which can be executed remotely over a network on a vulnerable machine. Consider a website running on a browser, assuming browser runs all websites in a single process, it can execute JavaScript code on the client. This JavaScript will use spectre variant 1 to train the branch predictor for speculative execution. Since the website is running in a single process, like a thread, it will use this to access an arbitrary memory location which may lead to leaking of data of other websites. This data can be passwords, authentication tokens, cookies, etc. This has been prevented by site isolation. Reis C, (2019).

Listing 1: Conditional Jump

```
        if (x < array1_size)
y = array2 [ array1 [ x ]*4096];
```

2.4 Microarchitectural attack: Meltdown

Meltdown exploits the out-or-order execution available on most of the modern processors Lipp M, (2018). This attack enables an adversary to read any arbitrary memory without having the privileges to do so. Out-of-order execution is a feature of the processor which reorders the micro-ops and execute out of their natural order as it was in the code. This feature is indispensable for performance in modern CPUs. For example, the CPU has to wait for the data to be fetched from the memory, so instead of idling, subsequent instructions can be executed. These instructions can then be later retired as per their actual order. Meltdown exploits this feature, and causes CPU to access any arbitrary memory to read sensitive data and leak it over the side channel. Unlike spectre, meltdown does not need tailoring as per the victim's software environment Lipp M, (2018).

To read arbitrary memory, the content of the inaccessible address is loaded in a register. An instruction is executed in out-of-order to access a cache line based on the value of the register. For example, an array is indexed based on the value available in the register, so that array location gets cached. When the instructions are retired it is observed that invalid memory access was done and the process is terminated. Flush+Reload side channel attack is used to read the data. The attacker employs exception handling to avoid crashing of the process and repeat the method to extract data. This attack can be used to extract all the kernel memory as the kernel addresses are directly mapped to a predefined virtual address in the user address space.

Meltdown can be mitigated by using Kernel Address Space Layout Randomization, but can be broken easily. KAISER, Lipp M, (2018) is a kernel modification which does not allow kernel memory mapped to user space mitigates the meltdown attack.

2.5 Microarchitectural Attack: Foreshadow

Intel SGX (Intel. Intel® 64 and IA-32 Architectures, (2018) is hardware extensions that provide a protected container for execution referred as enclave. These enclaves provide integrity and confidentiality even in the presence of privileged malware. These enclaves are managed by the processor itself and access to data is restricted to the code inside the enclave. No other software, OS or hypervisor is allowed to access these enclaves. Thus, these isolated execution environments ensure the security of the code being executed and its data. These enclaves also provide an attestation for enclaves to prove that application has not been compromised. Even though it is not possible to compromise the code directly or extract secret data, microarchitectural attacks have broken these enclaves to extract the sensitive data through side channels and meltdown type attack. Foreshadow Van Bulck J,et al. (2018) and Sgxpectre (Chen G, et al. (2018) are few of the latest examples of microarchitectural attacks on Intel SGX. Foreshadow uses meltdown type out-of-order execution to execute transient instructions which read secret data and bring it to the cache, its attack is divided into three phases.

2.6 Hardware Exploit: Backdoor In X86 Processor

Christopher Domas Domas C, (2018), in his paper discusses the existence of backdoor in VIA C3 processor. The processor has a hidden RISC core, undocumented, hidden from the users. The paper also discusses the procedure of its discovery. The hidden core also referred to as deeply embedded core, is activated by a model-specific register (MSR) which add new instruction set to existing x86 instruction set. This instruction set can be used to execute code directly on the hidden core bypassing the security features of the system to gain root privileges. The method for detecting hidden core to using it as a backdoor for privilege escalation has been simplified as follows:

1. Study of the architecture patents to discover the possible undisclosed architecture.

2. The possible hidden core can be activated by Global configuration register using MSR.

3. Searching the possible MSR used as Global configuration register from 1300 implemented MSRs, which is reduced to 43 as the functionally unique MSRs.

4. A total of 2752 bits can lead to different configurations of the processors. Invalid

configuration leads to system crash, so the number of bits to be searched is reduced.

5. Sandsifter, Github.com (2019), is a tool which is used to find the hidden launch instruction.

6. With launch instruction, the hidden instruction set was discovered, which is executed on hidden core.

3.0 REVIEW ON EXISTING HARDWARE SECURITY

Information flow tracking (IFT) has been widely used in the field of security for HW/SW systems. Its core idea is to enhance the hardware and/or the software in a way that the flow of information is explicitly visible. Different techniques have been proposed in order to instrument the processor with additional components enabling it to monitor information flows and possibly block the illegal flows at run time, S. Devadas, (2004). Also, CC-hunter, J. Chen and G. Venkataramani (2014) targets illegal information flows at run time.

Instruments a processor with a special module called covert channel auditor which uncovers covert channel communications between a trojan and a spy process through detecting specific patterns of conflicting accesses to a shared resource, J. Chen and G. Venkataramani (2014). All these methods incur high overhead on the design and demand modifications at different levels of the system, such as in the instruction set architecture (ISA), the operating system (OS) and the HW implementation. CC-hunter also requires defining the conflict indicator event for each resource which can open a covert channel. This demands a priori knowledge about possible covert channels. In order to capture and remove timing-based side channels in the design the gate-level netlist can be instrumented with IFT capabilities. Such a gate-level IFT method is meant to be applied to selected modules such as a crypto-core or a bus controller. It faces complexity problems when addressing security issues of the larger system. Moreover, since the underlying analysis is based on simulation, relevant corner cases may be missed. (Ardeshiricham et al.2017) developed an RTL transformation framework, called RTLIFT, for enhancing an existing RTL model with information flow tracking (IFT). The method is mainly based on gate-level IFT, S. Devadas, (2004). and has been extended by the Clepsydra approach to covering information flow.

Instruction level abstraction (ILA) is an approach to create a HW-dependent model of the software and formally verify security requirements, P. Subramanyan,(2016). This method has the advantage of capturing vulnerabilities which are not visible in the software/firmware itself, but rather manifest themselves through the communication between software and hardware. However, HW security vulnerabilities in the RTL implementation of the hardware cannot be detected by this approach. The use of formal techniques for security verification in hardware was pioneered in,

G.Cabodi, (2017) by adopting the idea of taint analysis which originally comes from the software domain. This is the research which is the most closely related to our work. In those approaches a HW security requirement such as a specific confidentiality requirement is formulated in terms of a taint property, P.Subramanyan (2016) along a certain path in the design. The property fails if the taint can propagate, i.e., the information can flow from the source of the path to its destination, while conforming to a propagation condition.

Pan et al, (2011), first present IVF of reorder buffer and register file considering different intermittent fault models, and then compute IVF by changing microarchitecture parameters and program phases. And finally introduce several IVF guided protection techniques to improve system reliability. In addition, Dweik et al (2014, March) in their simulation experiments, chose ten SPEC CPU2006 benchmarks and fast forwarded through the first 300 million instructions then ran detailed simulations for three billion instructions. And Fadiheh et al (2019, March) investigated general flow of UPEC-based hardware security analysis. Checking the UPEC property is at the core of a systematic, iterative process by which the designer identifies and qualifies possible hardware vulnerabilities in the design. The UPEC property is initialized on a bounded model of length dMEM and with a proof assumption and obligation for the complete set of microarchitectural state variables.

Hardware Trojans are designed by attackers to add unwanted functionality to the design. There is not a standard procedure to design hardware Trojans, as its design is reliant upon the attacker goals and available resources. Despite this, hardware security researchers have categorized different Trojans. For example, authors in, Jin, Y.; Makris, (2008) divided Trojans into implicit payload and explicit payload based on the Trojan's activities when triggered. Various Trojan designs were also proposed based on the stealth of hardware Trojans and the impact they may cause, King,S et al. (2008). While most of these Trojans are inserted at the register transfer level (RTL), Trojan insertion is also possible through dopant manipulation. As existing testing methods fall short in detecting malicious logic, dedicated hardware Trojan detection methods and trusted integrated circuit design have been developed in recent years, Karri, R. et al. (2010). A large body of hardware Trojan detection and prevention methods have been proposed, which can be divided into four main categories: (i) enhanced functional testing; (ii) side-channel fingerprinting (iii) Trojan prevention; and (iv) circuit hardening.

3.1 Mitigation Techniques for Meltdown and Spectre Attacks

The vulnerabilities attack at the hardware level and it cannot be patched or there is no specific fix for each of the vulnerabilities but there are techniques introduced by the vendors to mitigate the vulnerabilities. By mitigating means the vulnerabilities will be minimized or to prevent it from being severe but not vulnerabilities will not be removed completely. Many major processors vendor such as Intel, AMD and ARM have reported that the processors have being affected more than one variant of vulnerabilities. The expert around world and the processor vendors has been working around the vulnerabilities to research and product the effective techniques and patches to mitigate the attacks. Microsoft,

Apple, Google, Firefox and Samsung has released operating system patches for most version of their machine to prevent this attack from attacking their users, their patches was announced publicly at the starting of January 2018, Apple Inc. (2018).

The hardware attacks use side channels to extract the data, but it is the features of the processor that put this data over side channel. It can be observed that, these attacks are limited to data extraction only, which can be mitigated by taking proper measures. For example, spectre can be mitigated by isolation or by removing the observation channels Swiat, (2019). As existing hardware is not as easy to patch, there is a need for software workarounds until new hardware can be deployed. Gruss et al (2017)] proposed KAISER, a kernel modification to not have the kernel mapped in the user space. This modification was intended to prevent side-channel attacks breaking KASLR (HUND,R, 2013). However, it also prevents Meltdown, as it ensures that there is no valid mapping to kernel space or physical memory available in user space

SUMMARY AND FUTURE WORK

Trojan detection methods have been successful in detecting certain hardware Trojans, the scope of detection is limited because they rely on over-simplified, sometimes erroneous assumptions including: use traditional and simple circuit structures which limit their functionality; attempt to occupy negligible on-chip area in order to mask the Trojan profile from the overall side-channel profile. All these assumptions are widely embraced by the hardware security community. Unfortunately, these assumptions are proving invalid, and it is becoming more apparent that the problem of hardware security and hardware Trojan detection is more complicated and broader than what we once imagined.

Future work; to overcome the shortage of this method, a real-time trust evaluation structure is proposed that can constantly monitor the operational status of the target circuit and report circuit abnormalities instantly. Concurrent on-chip parity checking leveraging on resistive RAM is also developed for run-time hardware Trojan detection.

It has shown that covert channel attacks are not limited to high-end processors but can affect a larger range of architectures. While all previous attacks were found by clever thinking of a human attacker, Mohammad Rahmani Fadiheh et al.(2019) presented UPEC, an automated method to systematically detect all vulnerabilities by covert channels, including those by covert channels unknown so far.

Future work; will explore measures to improve the scalability of UPEC to handle larger processors. By automating the construction of induction proofs described in Sec. VI, the UPEC computational model can be restricted to only two clock cycles, thus, drastically reducing computational complexity. In addition, a compositional approach to UPEC will be explored.

CONCLUSION

This paper, discussed the different ways to bypass the traditional security mechanisms using nontraditional ways. It is extremely difficult to observe these kind attacks on the system as it does not exploit software but use the necessary features provided by the underlying hardware. The operations such as out-of-order execution, speculative execution, are necessary features of modern CPUs, but these operations are completely hidden from upper layers such as OS. The end result is the expected outcome, but the side effects can be exploited. This makes the study of microarchitectural attacks very difficult at the same time also makes it difficult to exploit the target since direct observation is not possible. But a successful exploit will lead to a serious security threat very well hidden from security mechanisms.

REFERENCES

1. Ardeshiricham, W. Hu, J. Marxen, and R. Kastner, “Register transfer level information flow tracking for provably secure hardware design,” in Design, Automation & Test in Europe Conference (DATE). IEEE,2017, pp. 1695–1700.
2. Anonymous. The proof-of-concept code for Spectre attacks: exploiting speculative execution. 2018. https://gist.github.com/anonymous/ 99a72c9c1003f8ae0707b4927ec1bd8a. Accessed January 1, 2019.
3. Apple Inc. (2018). About speculative execution vulnerabilities in ARM-based and Intel CPUs. https://support.apple.com/enmy/HT208394
4. Chen G, Chen S, Xiao Y, Zhang Y, Lin Z, Lai TH. Sgxpectre attacks: stealing intel secrets from sgx enclaves via speculative execution. arXiv preprint arXiv:1802.09085. 2018.
5. Constantinescu, C. “Intermittent faults and effects on reliability of integrated circuits”. In *2008 Annual Reliability and*

Maintainability Symposium (pp. 370-374). IEEE. (2008, January).

6. D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, D. Ponomarev, et al. BranchScope: A New Side-Channel Attack on Directional Branch Predictor. ACM Conference on Architectural Support for Programming Languages and Operating Systems, pages 693–707, 2018.
7. Domas C. Breaking the x86 ISA. Black Hat, USA; 2017. https://www.blackhat.com/us-17/briefings.html#christopher-domas.
8. Domas C. Hardware backdoors in x86 CPUs. 2018. https://www.blackhat.com/us-18/briefings/schedule/index.html#god-mode-unlocked-hardware-backdoors-in-x-cpus-10194.
9. Fatemeh Soleimani Roozbahani and Reihaneh Azad "Security Solutions against Computer Networks Threats" Int. J. Advanced Networking and Applications Volume: 07 Issue: 01 Pages: 2576-2581 (2015) ISSN: 0975-0290
10. G.Cabodi, P. Camurati, S. F. Finocchiaro, F. Savarese, and D. Vendraminetto, "Embedded systems secure path verification at the HW/SW interface," IEEE Design & Test, vol. 34, no. 5, pp. 38–46, 2017.
11. G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," in ACM Sigplan Notices, vol. 39, no. 11. ACM, 2004, pp. 85–96.
12. Github.com. xoreaxeaxeax. SandSifter. 2017. https://github.com/xoreaxeaxeax/sandsifter. Accessed January 2, 2019.
13. 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.
14. Hund,r., willems, c., and holz, T. Practical Timing Side Channel Attacks against Kernel Space ASLR. In S&P (2013)
15. Intel. Intel® 64 and IA-32 Architectures Software Developer's Manual. Intel; 2018. https://software.intel.com/en-us/download/intel-64-and-ia32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.
16. J. Chen and G. Venkataramani, "CC-Hunter: Uncovering covert timing channels on shared processor hardware," in Annual IEEE/ACM Intl. Symp. on Microarchitecture. IEEE, 2014, pp. 216–228.
17. Jin, Y.; Makris, Y. Hardware Trojan Detection Using Path Delay Fingerprint. In Proceedings of the IEEE International Workshop on Hardware-Oriented Security and Trust, Anaheim, CA, USA, 9 June 2008; pp. 51–57.
18. J. Van Bulck, F. Piessens, and R. Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. USENIX Security Symposium, 2018.
19. King, S.; Tucek, J.; Cozzie, A.; Grier, C.; Jiang, W.; Zhou, Y. Designing and Implementing Malicious Hardware. In Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET), San Francisco, CA, USA, 15 April 2008; pp. 1–8.
20. Kocher P, Horn J, Fogh A, et al. Spectre attacks: exploiting speculative execution. Paper presented at: RSA Conference 2018; 2019; San Francisco,CA.
21. Lipp M, Schwarz M, Gruss D, et al. Meltdown: reading kernel memory from user space. Paper presented at: Proceedings of the 27th USENIX Security Symposium; August 15-17, 2018; Baltimore, MD.
22. Liu F, Yarom Y, Ge Q, Heiser G, Lee RB. Last-level cache side-channel attacks are practical. Paper presented at: 2015 IEEE

Symposium on Security and Privacy, San Jose, CA: IEEE; 2015: 605–622.

23. M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," arXiv preprint arXiv:1801.01207, 2018.
24. Mohammad Rahmani Fadiheh, Dominik Stoffel, Clark Barrett, Subhasish Mitra, Wolfgang Kunz ''Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking'' arXiv:1812.04975v1 [cs.CR] 5 Dec 2018
25. M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in ACM Sigplan Notices, vol. 44, no. 3. ACM, 2009, pp. 109–120.
26. O. Aciicmez and J.-P. Seifert, "Cheap hardware parallelism implies cheap security," in Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC). IEEE, 2007, pp. 80–91.
27. P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp,S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," arXiv preprint arXiv:1801.01203, 2018.
28. P. Pessl, D. Gruss, C. Maurice, and S. Mangard, "Reverse engineering intel DRAM addressing and exploitation," ArXiv e-prints, 2015.
29. P. Subramanyan, S. Malik, H. Khattri, A. Maiti, and J. Fung, "Verifying information flow properties of firmware using symbolic execution," in Design, Automation & Test in Europe Conference (DATE). IEEE, 2016, pp. 337–342.
30. Reis C. Mitigating spectre with site isolation in chrome. 2018. https://security.googleblog.com/ 2018/07/mitigating-spectre-with-site-isolation.html. Accessed January 4, 2019.
31. Swiat. Mitigating speculative execution side channel hardware vulnerabilities. 2018. https://blogs.technet.microsoft.com/srd/2018/ 03/15/ mitigating-speculative-execution-side-channel-hardware-vulnerabilities. Accessed January 4, 2019.
32. Vaibhav G. Lokhande Deepti Vidyarthi,''A study of hardware architecture-based attacks to bypass operating system security'' WILEY, Published on: 11 June 2019
33. Van Bulck J, Minkin M, Weisse O, et al. Foreshadow: extracting the keys to the Intel SGX kingdom with transient out-of-order execution. Paper presented at: 27th USENIX Security Symposium; 2018.
34. Weisse O, Van Bulck J, Minkin M, et al. Foreshadow-NG: breaking the virtual memory abstraction with transient out-of-order execution. Paper presented at: USENIX Security Symposium, Technical report. 2018. https://foreshadowattack.eu/.
35. YaromY,FalknerK.FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. Paper presented at: 23rd USENIX Security Symposium; August 20–22, 2014; San Diego, CA: 2014: 719–732.
36. YaromY,FalknerK.FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. Paper presented at: 23rd USENIX Security Symposium; August 20–22, 2014; San Diego, CA: 2014: 719–732.
37. Y. Yarom, D. Genkin, and N. Heninger, "Cachebleed: a timing attack on OpenSSL constant-time RSA," Journal of Cryptographic Engineering, vol. 7, no. 2, pp. 99–112, 2017.