Timing the Transient Execution: A New Side-Channel Attack on Intel CPUs

Yu Jin*, Pengfei Qiu*, Chunlu Wang†, Yihao Yang‡,
Dongsheng Wang§, Gang Qu¶

*†‡Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education
§Tsinghua University ¶University of Maryland
lambda.jinyu@gmail.com, {qpf, wangcl}@bupt.edu.cn, khaosyg@gmail.com,
wds@tsinghua.edu.cn, gangqu@umd.edu

***Abstract*—The transient execution attack is a type of attack leveraging the vulnerability of modern CPU optimization technologies. New attacks surface rapidly. The side-channel is a key part of transient execution attacks to leak data.**

In this work, we discover a vulnerability that the change of the EFLAGS register in transient execution may have a side effect on the Jcc (jump on condition code) instruction after it in Intel CPUs. Based on our discovery, we propose a new side-channel attack that leverages the timing of both transient execution and Jcc instructions to deliver data. This attack encodes secret data to the change of register which makes the execution time of context slightly slower, which can be measured by the attacker to decode data. This attack doesn't rely on the cache system and doesn't need to reset the EFLAGS register manually to its initial state before the attack, which may make it more difficult to detect or mitigate. We implemented this side-channel on machines with Intel Core i7-6700, i7-7700, and i9-10980XE CPUs. In the first two processors, we combined it as the side-channel of the Meltdown attack, which could achieve 100% success leaking rate. We evaluate and discuss potential defenses against the attack. Our contributions include discovering security vulnerabilities in the implementation of Jcc instructions and EFLAGS register and proposing a new side-channel attack that does not rely on the cache system.

***Index Terms*—Timing, EFLAGS register, side-channel attacks**

I. Introduction

The increasing complexity and aggressive optimizations of modern CPUs, with their many microarchitectural features, have led to improved performance, but they have also created a range of security vulnerabilities [1], [2]. This complexity and optimization are the root cause of many security issues, including side-channel attacks [3], Meltdown attack [4], [5], Spectre attack [6], [7], Microarchitectural data sampling (MDS) attack [8]–[10], fault injection attack [11]–[16], and more. The complex and dynamic nature of modern CPUs has made them a challenging target for security researchers and developers to discover and mitigate, and a constant source of concern for users. As the field of computer security continues to evolve, new techniques and countermeasures will be needed to keep pace with the ever-evolving threat landscape.

*Equal contributions.

A number of the microarchitectural side-channel attacks are based on the side effects and state of the cache system. There are many ways to leak info through the cache system. After the first cache-timing attack reported by Bernstein in 2005 [17]. Multiple variations such as evict+time (2006) [18], flush+reload(2014) [19], prime+probe(2015) [3], flush+flush(2016) [20]. New cache side-channel attacks are still being discovered, e.g., attacks in cache replacement policies(2020) [21], streamline(2021) [22], using cache dirty states(2022) [23], cache coherence(2022) [24]. Besides, some attacks do not directly rely on the cache system, such as PortSmash [25], PlatyPus(2021) [26], PMU-Spill [27].

Transient execution attacks [28], including Meltdown, Spectre, and MDS attacks, exploit the complex and aggressive optimizations [10] of modern CPUs to leak sensitive information through transient states. As a high-level overview [28], transient execution attacks consist of five phases: (1) microarchitectural preparation, (2) triggering a fault, (3) encoding secret data to a covert channel, (4) flushing transient instructions, and (5) decoding the secret data. The success of phases 3 and 4 depends on side-channels, which require the channel state to be initialized or set to a specific state. For example, in the flush+reload attack [19], the attacker needs to flush the monitored memory line from the cache in phase 1 and encode secret data by loading one index of the memory line into the cache in phase 4. This allows the attacker to measure the time of the monitored memory line being loaded in the cache in phase 5 to decode data.

Reverse engineering efforts, such as those by uops.info [29], have attempted to reveal information about the behavior of a processor's microarchitecture despite the lack of publicly available implementation details. Intel's manual [30] has provided additional insights into instruction performance characteristics, highlighting that certain instructions can cause pipeline stalls or other effects due to their functional requirements [31]. For example, the `MFENCE` instruction introduces a stall in the pipeline until all previous memory operations have been completed. Awareness of these nuances is crucial for optimizing code to maximize performance on a particular processor architecture and identify potential security vulnerabilities.

In our work, we conduct in-depth research on the behavior

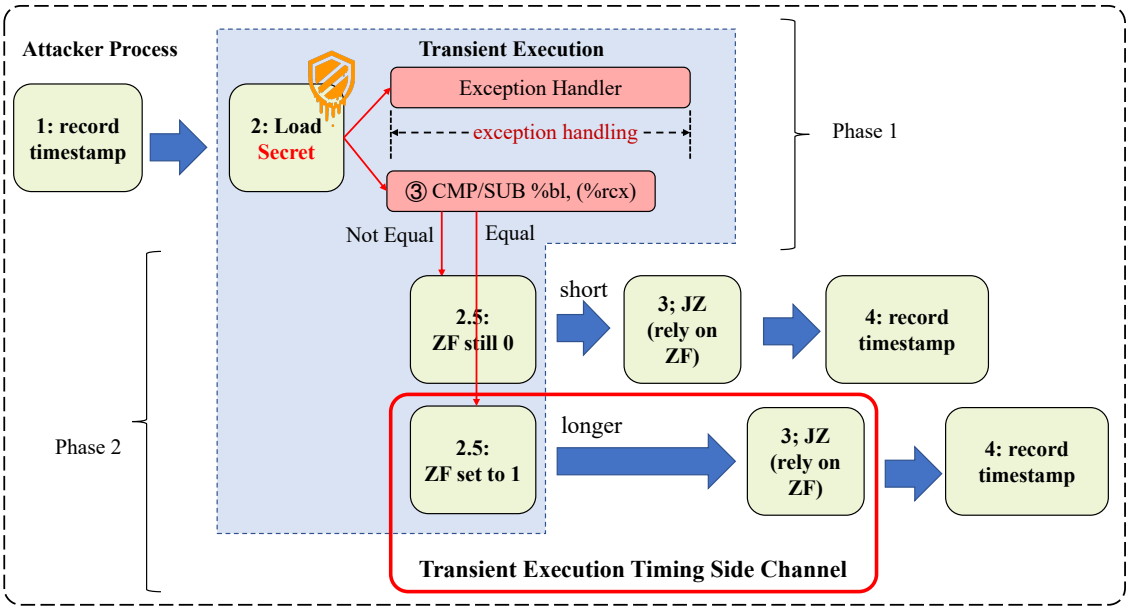

Fig. 1: Overview of Transient Execution Timing Side-Channel.

and side effects of transient execution attacks and discover a vulnerability of the implementation in Intel CPUs. Specifically, the change of the EFLAGS register in transient execution may influence the Jcc instruction after it. Based on our discovery, we introduce a novel side-channel attack that leverages the timing of transient execution with Jcc instruction. The change of EFLAGS in transient execution could make some Jcc instructions after it slightly slower. As we showed in Fig.1, by encoding secret data to the EFLAGS register, we can measure the execution time of the Jcc instruction's context to decode data without the need to reset the EFLAGS register to its initial state in phase 1 of transient attack.

This attack does not rely on the cache system, which may make it more difficult to detect compared to previous side-channel attacks. We implement this side-channel in real machines with Intel Core i7-6700 and i7-7700 and i9-10980XE CPUs. We build the Meltdown attack with our side-channel attack and evaluate it on i7-6700 and i7-7700. In practice, our side-channel can achieve 100% success rate.

To mitigate this attack, we propose several practical mitigation methods based on our evaluation.

Our research makes several contributions:

1) We discover security vulnerabilities in the implementation of EFLAGS register and Jcc Instruction. The change of EFLAGS register during transient execution can affect the timing of Jcc instruction after it.
2) We propose a novel side-channel attack that exploits the timing of execution affected by Jcc instructions, which are dependent on the EFLAGS register. Our attack is distinct from previous side-channel attacks appearing in that it does not rely on the cache system and does not require resetting the initial state during the preparation phase of the transient execution attack.
3) We implement this side-channel in Intel Core i7-6700 and i7-7700 and i9-10980XE CPUs. We build the Meltdown attack with this side-channel on Intel Core i7-6700 and i7-7700 CPUs on a real machine and it could achieve 100% success rate.

To the best of our knowledge, this is the first time that the EFLAGS register has been used as a side-channel. We hope that our work can help to improve the security of future CPUs and bring insight for microarchitecture attack research. The source code of our attacks would be published at https://github.com/.

The rest of the paper is organized as follows. In Section II, we introduce the background knowledge of side-channel attacks and transient execution attacks. In Section III, we present the details of the attack. In Section IV, we evaluate our attack on Intel CPU. In Section V, we propose several mitigations for this side-channel. In Section VI, we discuss the limitation of this attack and future work. Finally, we conclude our work in Section VII.

II. Background

A. Microarchitecture

The microarchitecture [32] of each core of the CPU is composed of several components, such as the cache system, the frontend that includes the branch predictor, the out-of-order execution unit, etc. The CPU core is the core of the CPU, which is responsible for the execution of the instructions. The cache system is used to store the data and instructions that are frequently accessed. The frontend is responsible for the instruction fetch and decode. The out-of-order execution unit is responsible for the out-of-order execution of the instructions. As the modern CPU is an microarchitecture complex system. For most commercial CPUs, the microarchitecture of the CPU is a black box, and much research trying to reverse engineering in it [33]–[35].

B. Side-Channel Attacks

Side-Channel Attacks in microarchitecture [36] is a class of attacks that exploit the side effects of a program to leak information about the program's execution. The side effects can be the cache system [3], [17]–[21], [37], the branch predictor [38], [39], the power [26], etc. For example, the cache system can be used to leak info about the memory access pattern of the program. The branch predictor can be used to leak info about the control flow of the program. The root cause [1] of most of the side-channel attacks in microarchitecture is the shared resource, which is one key to the performance optimization of the CPU.

C. Transient Execution Attacks

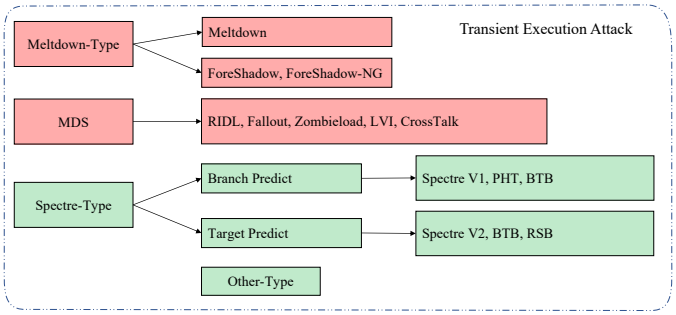

Fig. 2: Category of Transient Execution Attacks

Since Meltdown [4] and Spectre [6] attacks have been discovered, the transient execution attacks [28] has been a hot topic in the security community. The transient execution can be caused by the fault, the branch misprediction, the cache miss, etc. And there are several variations of transient execution attacks [5], [7]–[10], [40]–[44]. IP vendors like Intel and AMD have released the microcode update to mitigate the transient execution attacks [45]. The researcher also proposed some countermeasures to mitigate it [1], [28], [46]–[48].

III. Transient Execution Timing Attacks

A. Assumption and Threat Model

Assumption: The secret data can be accessed through a transient execution attack. There are lots of transient execution attacks such as Meltdown [4], Spectre [6], Foreshadow [5], ZombieLoad [10], etc. Although most existing attacks have been mitigated, they may have undisclosed transient execution vulnerabilities in the CPUs and have been exploited in the future [27].

Threat Model: The attacker runs in the unprivileged mode. And there is another victim process that runs on the same machine. We define the threat model of this study as: the attacker utilizes transient execution timing to recover the secret data acquired in the transient execution attacks.

B. Attack Overview

We implement the attack as the side-channel of the Meltdown attack, shown in Fig. 1. The attack is composed of two phases. In the first phase, we trigger transient execution and encode the secret data through the EFLAGS register. In the second phase, we measure the execution time of the Jcc instruction's context to decode data. To encode a secret through a binary flag, we need to use iteration `test_num` to set the flag. If the `test_num` equals the secret, the flag will be set and the secret would be encoded successfully.

C. Implement Detail

We notate the `secret_addr` as the address of the secret data. And the `offset` is the offset of the `secret_addr`. The `EFLAGS instruction` is the instruction that can change the EFLAGS register. The `Jcc instruction` is the instruction that can be influenced by the EFLAGS register. The available instruction set is the list in Table I. We use `__rdtsc` from `x86intrin.h` to get the time-stamp counter of the CPU.

```
for (uint8_t test_num = 0; test_num <= TO;
     test_num++){
     start_time = __rdtsc();
     // timing context start
     asm volatile(
          "MOV %0, %%RCX;"
          "MOV %1, %%BL;"
          :
          : "r"(secret_addr + offset),
            "r"(test_num)
          :);
     if (xbegin() == (~0u))
     {
          // EFLAGS instruction
          asm volatile("SUB %BL, (%RCX);");
     }
     asm volatile(
          "JZ equal;" // Jcc instruction
          "JMP notequal;"
          "equal: NOP;"
          "notequal: NOP;");
     // timing context end
     spend_time = __rdtsc() - start_time;
     if (max_time < spend_time)
     {
          max_time = spend_time;
          argmax = i;
     }
}
```

Listing 1: Pseudocode for timing the transient execution attack in Intel X86 architecture.

The `secret_addr` is an address in kernel space. And the attacker running in an unprivileged mode which can not access the `secret_addr`. The `offset` is the offset of the `secret_addr`. In the TSX transaction, the attacker will try to access the `secret_addr` by `sub` instruction. A transient execution will be caused by the fault. During the transient execution, the `ZF` may be set to 1 if the secret data in `*(secret_addr+offest)` is equal to `i`. And the `ZF` will be restored to 0 after the transient execution. The `ZF` will be used by the `JZ` instruction to determine whether to jump to the `equal` label or the `notequal` label. In our experiment, if the secret data in `*(secret_addr+offest)`

is equal to `i`, the execution time of the context will be slower than the execution time of the context that the secret data in `*(secret_addr+offest)` is not equal to `i`. The `max_time` is the maximum execution time of the context. The `argmax` is the secret data, as we have shown the distribution in Fig. 3. The `max_time` and `argmax` can be used to decode the secret data.

TABLE I: Instruction for our Attack.

Type	Instruction	Description	EFLAGS
EFLAGS	SUB	Subtract.	ZF
	CMP	Compare Two Operands.	ZF
	CMPXCHG	Compare and Exchange.	ZF
Jcc	JE	Jump short if equal (ZF = 1).	ZF
	JZ	Jump short if zero (ZF = 1).	ZF

IV. Experiment and Evaluation

A. Experimental Setup and Result

We implement the attack in Intel i7-6700, i7-7700, and i9-10980XE CPU. The experiment is running in the Ubuntu 16.04 xenial with kernel version 4.15.0 (i7-6700, i7-7700) and Ubuntu 22.04 jammy with kernel version 5.15.0 (i9-10980XE). We build the Meltdown attack with our side-channel to read kernel memory from user space and achieve 100% success leaking rate in the first two processors.

B. Evaluation

In our experiment, we found that the influence of the EFLAGS register on the execution time of Jcc instruction is not as persistent as the cache state. For about 6-9 cycles after the transient execute, the Jcc execute time will not be about to construct a side-channel.

Empirically, the attack needs to repeat thousands of times for higher accuracy. For the Listing1, we use Intel TSX, a transactional memory implementation, to completely suppress the exception. We also use system interrupt handlers to suppress the exception and achieve the same effect. The TSX is more efficient than the system interrupt handlers for transient execution attacks.

Though the statistical distribution of the argmax of timing can be used to decode the secret data. The distribution of the average clock, due to the noise, cannot be used as a side-channel, as we have shown in Fig. 4.

V. Mitigation

For mitigating the Transient Execution Timing attacks, we can use two gadgets to delay the Jcc instruction or rewrite the EFLAGS register after the transient execution.

A. Hareware Mitigation

The implementation of the Jcc instruction should not have timing or other side effects on different conditions to avoid the adversarial execution measuring.

B. Delay Jcc

If the Jcc instruction is not executed immediately after the EFLAGS register has been changed, the influence of the EFLAGS register can be reduced. 10 cycles are enough to reduce the influence of the EFLAGS register. `.rept count` Repeat the sequence of lines between the `.rept` directive and the next `.endr` directive count times. By using the `NOP` instruction to delay the Jcc instruction, we can reduce the influence of the EFLAGS register on the execution time. There are lots of ways to delay, We just give one example here.

C. Rewrite EFLAGS

The `LAHF` and `SAHF` instructions are x86 assembly language instructions that are used to manipulate the low 8 bits of the FLAGS register in the x86 processor [30]. The `LAHF` instruction is short for "Load AH from Flags". It loads the low 8 bits of the FLAGS register into the AH register while leaving the upper 8 bits of the AH register unchanged. The AH register is a 16-bit register that is used to store the high byte of the AX register. The `SAHF` instruction is short for "Store AH into Flags". It stores the low 8 bits of the AH register into the low 8 bits of the FLAGS register while leaving the upper 8 bits of the FLAGS register unchanged.

The `PUSHF` and `POPF` instructions are x86 assembly language instructions that are used to push and pop the contents of the FLAGS register onto and off of the stack, respectively. The `PUSHF` instruction pushes the entire 16-bit FLAGS register onto the top of the stack. This includes the status flags that are used to indicate the result of arithmetic and logic operations, as well as other control flags that control the behavior of the processor. The `POPF` instruction pops the contents of the top of the stack into the FLAGS register. This can be used to restore the state of the FLAGS register after it has been saved by a previous `PUSHF` instruction.

By rewriting the EFLAGS though `LAHF` and `SAHF`, or `PUSHF` and `POPF` instructions, the influence of the EFLAGS register can be reduced.

VI. Discussion and Futrue Work

The root causes of this attack are still not fully understood. We guess that there is some buffer in the execution unit of the Intel CPU which need some time to revert if the execution should be withdrawn. This withdrawal process will cause a stall if the following instruction depends on the target of the buffer.

A. Limitation

This timing attack relies on other transient execution attacks to build a real-world attack and it is easy to be disturbed by noise. But it is still a new side-channel attack and worth further exploration. This attack may bring insight for new microarchitecture attacks and give a new way to build side-channel attacks in cache side-channel resistant CPU.

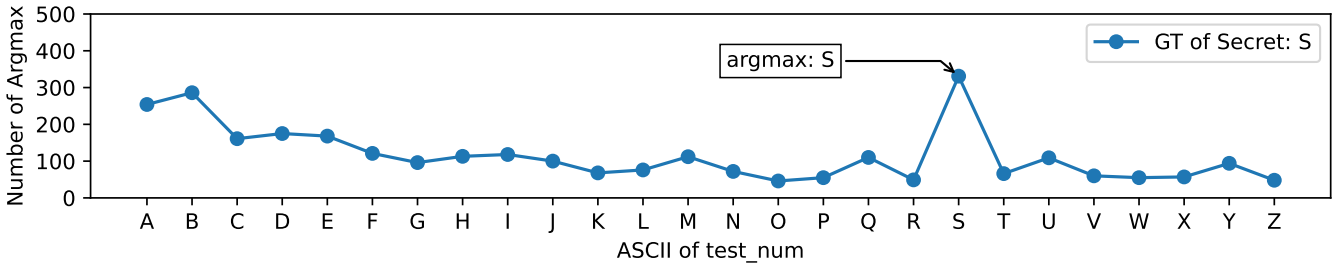

Fig. 3: Distribution of argmax.

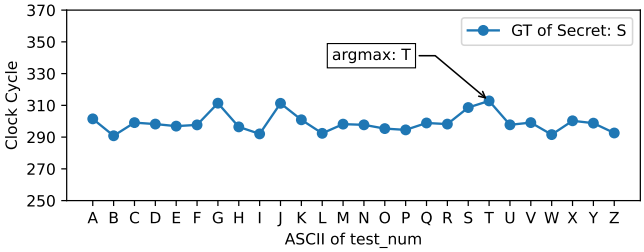

Fig. 4: Distribution of average clock.

```
asm volatile(
    ".rept 6 \n\t"
    "NOP \n\t"
    ".endr \n\t");
```

Listing 2: Pseudocode 1 for mitigating Transient Execution Timing attacks in Intel X86 architecture.

B. Futrue Work

a) Combined with other Transient Execution Attacks: More experiments are underway to fully understand it. Which will be published in the future.

b) Finding other various: There may be other microarchitectural components and instructions that can be used as timing side-channels. We will continue to explore this area.

VII. Conclusions

We present a new side-channel attack that leaks info through the timing of execution. When the ZF has been changed from 0 to 1 during transient execution caused by a fault in the Meltdown attack, though the ZF will be restored to 0 after the transient execution, the instruction like JZ execution time will be slightly longer. As a result, we can leak info throw EFLAGS register by measuring the execution time of context. Compared with previous side-channel attacks, our attack does not rely on the cache system, which may make it difficult to be detected by existing tools or methods [49]–[51].

As far as we know, our work is the first that build side-channel with the EFLAGS register. Hope our work can help to improve the security of future CPUs.

```
asm volatile(
    "LAHF;"
    "SAHF;"
    // or
    "PUSHF;"
    "POPF;" );
```

Listing 3: Pseudocode 2 for mitigating Adversarial Jcc attacks in Intel X86 architecture.

References

[1] N. R. Holtryd, M. Manivannan, and P. Stenström, "Sok: Analysis of root causes and defense strategies for attacks on microarchitectural optimizations," *arXiv preprint arXiv:2212.10221*, 2022.

[2] Intel, "Analysis of Speculative Execution Side Channels," https://software.intel.com/sites/default/files/managed/b9/f9/336983-Intel-Analysis-of-Speculative-Execution-Side-Channels-White-Paper.pdf, 2018.

[3] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE Symposium on Security and Privacy*, 2015.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in *Usenix Security*, 2018.

[5] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," in *Usenix Security*, 2018.

[6] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *S&P*, 2019.

[7] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre Returns! Speculation Attacks using the Return Stack Buffer," in *WOOT*, 2018.

[8] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: Rogue In-flight Data Load," in *S&P*. IEEE, 2019.

[9] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout: Leaking data on Meltdown-resistant CPUs," in *CCS*, 2019.

[10] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad : Cross-privilege-boundary data sampling," in *CCS*, 2019.

[11] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss, and F. Piessens, "Plundervolt: Software-based fault injection attacks against intel sgx," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 1466–1482.

[12] A. Tang, S. Sethumadhavan, and S. Stolfo, "{CLKSCREW}: Exposing the perils of {Security-Oblivious} energy management," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 1057–1074.

[13] P. Qiu, D. Wang, Y. Lyu, and G. Qu, "Voltjockey: Breaking sgx by software-controlled voltage-induced hardware faults," in *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2019, pp. 1–6.
[14] ——, "Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 195–209.
[15] P. Qiu, D. Wang, Y. Lyu, R. Tian, C. Wang, and G. Qu, "Voltjockey: A new dynamic voltage scaling-based fault injection attack on intel sgx," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 6, pp. 1130–1143, 2020.
[16] Z. Kenjar, T. Frassetto, D. Gens, M. Franz, and A.-R. Sadeghi, "{V0LTpwn}: Attacking x86 processor integrity from software," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1445–1461.
[17] D. J. Bernstein, "Cache-timing attacks on AES," 2005.
[18] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *CT-RSA*, 2006.
[19] Y. Yarom and K. Falkner, "Flush+Reload: A High Resolution, Low Noise, L3 Cache Side-channel Attack," in *Usenix Security*, 2014.
[20] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: a fast and stealthy cache attack," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2016, pp. 279–299.
[21] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "RELOAD+ REFRESH: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1967–1984.
[22] G. Saileshwar, C. W. Fletcher, and M. Qureshi, "Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 1077–1090.
[23] Y. Cui and X. Cheng, "Abusing cache line dirty states to leak information in commercial processors," in *2022 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2022.
[24] Y. Guo, A. Zigerelli, Y. Zhang, and J. Yang, "Adversarial prefetch: New cross-core cache side channel attacks," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 1458–1473.
[25] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García, and N. Tuveri, "Port contention for fun and profit," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 870–887.
[26] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 355–371.
[27] P. Qiu, Q. Gao, D. Wang, Y. Lyu, C. Liu, X. Li, C. Wang, and G. Qu, "Pmu-spill: Performance monitor unit counters leak secrets in transient executions," in *2022 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2022, pp. 1–6.
[28] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. Von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A systematic evaluation of transient execution attacks and defenses," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019.
[29] A. Abel and J. Reineke, "uops.info: Characterizing latency, throughput, and port usage of instructions on Intel microarchitectures," in *ASPLOS*, 2019.
[30] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, 2019.
[31] U. Degenbaev, "Formal Specification of the x86 Instruction Set Architecture," Ph.D. dissertation, Universität des Saarlandes, 2012.
[32] WikiChip, "Skylake (client) - Microarchitectures - Intel," https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client), 2018, accessed: May, 2021.
[33] B. Gras, C. Giuffrida, M. Kurth, H. Bos, and K. Razavi, "ABSynthe: Automatic Blackbox Side-channel Synthesis on Commodity Microarchitectures," in *NDSS*, 2020.
[34] D. Weber, A. Ibrahim, H. Nemati, M. Schwarz, and C. Rossow, "Osiris: Automated discovery of microarchitectural side channels," in *Usenix Security*, 2021.
[35] D. Lustig, M. Pellauer, and M. Martonosi, "Pipecheck: Specifying and verifying microarchitectural enforcement of memory consistency models," in *MICRO*, 2014.
[36] J. R. S. Vicarte, P. Shome, N. Nayak, C. Trippel, A. Morrison, D. Kohlbrenner, and C. W. Fletcher, "Opening Pandora's Box: A Systematic Study of New Ways Microarchitecture Can Leak Private Data," in *ISCA*, 2021.
[37] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 955–972.
[38] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18, 2018, pp. 693–707.
[39] A. R. H. Coronado, W. Lee, and W.-M. Lin, "Branchboozle: a side-channel within a hidden pattern history table of modern branch prediction units." in *SAC*, 2021, pp. 1617–1625.
[40] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, F. Piessens, and K. Leuven, "LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection," in *S&P*, 2020.
[41] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "CrossTalk: Speculative Data Leaks Across Cores Are Real," in *S&P*. IEEE, 2021.
[42] G. Maisuradze and C. Rossow, "ret2spec: Speculative Execution Using Return Stack Buffers," in *CCS*, 2018.
[43] P. Z. Google, "Speculative Execution, Variant 4: Speculative Store Bypass," https://bugs.chromium.org/p/project-zero/issues/detail?id=1528, 2018, accessed: May, 2021.
[44] V. Kiriansky and C. Waldspurger, "Speculative Buffer Overflows: Attacks and Defenses," *arXiv*, 2018.
[45] Intel, "Affected Processors: Transient Execution Attacks & Related Security Issues by CPU," https://www.intel.com/content/www/us/en/developer/topic-technology/software-security-guidance/processors-affected-consolidated-product-cpu-model.html, 2022, accessed: August, 2022.
[46] S. Cauligi, C. Disselkoen, D. Moghimi, G. Barthe, and D. Stefan, "SoK: Practical Foundations for Spectre Defenses," in *S&P*. IEEE, 2022.
[47] M. F. A. Kadir, J. K. Wong, F. Ab Wahab, A. F. A. A. Bharun, M. A. Mohamed, and A. H. Zakaria, "Retpoline technique for mitigating spectre attack," in *2019 6th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, 2019, pp. 96–101.
[48] A. Gonzalez, B. Korpan, J. Zhao, E. Younis, and K. Asanović, "Replicating and Mitigating Spectre Attacks on an Open Source RISC-V Microarchitecture," in *CARRV*, 2019.
[49] C. Li and J.-L. Gaudiot, "Detecting spectre attacks using hardware performance counters," *IEEE Transactions on Computers*, 2022.
[50] G. Wang, S. Chattopadhyay, A. K. Biswas, T. Mitra, and A. Roychoudhury, "KLEESpectre: Detecting information leakage through speculative cache attacks via symbolic execution," *TOSEM*, 2020.
[51] M. Guarnieri, B. Köpf, J. Morales, J. Reineke, and A. Sanchez, "Spectector: Principled Detection of Speculative Information Flows," in *S&P*. IEEE, 2020.

APPENDIX

Appendix A: Victim

The victim code is shown in Fig. 4, as same as the POC in Meltdown [4]. We run it parallelly with the attacker in the same physical core but a different logical core for a higher reading rate. The victim will try to keep the secret string cached in the cache line.

```
char *strings[] = {SECRET_STR};

while (1)
{
    // keep string cached for better
    results
    volatile size_t dummy = 0, i;
    for (i = 0; i < len; i++)
    {
        dummy += secret[i];
    }
    sched_yield();
}
```

Listing 4: Code Snippet for victim.