Whisper: Timing the Transient Execution to Leak Secrets and Break KASLR

[1]Yu Jin, [1]Chunlu Wang, [1,3]Pengfei Qiu*, [2]Chang Liu*, [1]Yihao Yang, [2]Hongpei Zheng
[2]Yongqiang Lyu, [1]Xiaoyong Li, [4]Gang Qu and [2,3]Dongsheng Wang
[1]Key Laboratory of Trustworthy Distributed Computing and Service (BUPT), Ministry of Education
[2]Tsinghua University, [3]Zhongguancun Laboratory, [4]University of Maryland, College Park

ABSTRACT

The vulnerabilities of transient execution have been exploited in many side-channel attacks (SCA). We report Whisper, a novel transient execution timing (TET) side channel, which is based on the execution time difference of transient execution under different conditions. We develop TET version of SCAs including Meltdown, Zombieload, and Spectre-RSB that use Whisper as covert channel to leak information. We further propose TET-KASLR to break the kernel address space layout randomization (KASLR) mechanism under the protection of KPTI and FLARE. These attacks are simple to implement and can bypass the existing mitigation methods because the TET side channel relies on execution time that can be conveniently obtained by architectural level timing analysis. We demonstrate the correctness and effectiveness of these attacks on various x86-64 CPUs. The root cause of Whisper is analyzed with our toolset built on performance monitor unit (PMU) and potential defense against Whisper is also discussed.

1 INTRODUCTION

Out-of-order execution and speculative execution have been widely deployed in modern processors, in which some instructions may not be committed due to the misprediction or exception caused by the proceeding instructions. The execution of these non-committed instructions, which is known as transient execution, cannot change the architecture state, but may leave a record on the cache or other microarchitectural resources. Numerous attacks such as Meltdown and Spectre have exploited this feature to leak secret data [13, 14, 17, 19, 21]. The cache within the memory subsystem is the most commonly exploited microarchitectural resource to transmit secrets fetched during transient execution, and several mitigation strategies against transient execution attacks concentrate on preventing the cache from leaking secrets [25, 27]. To circumvent these strategies, researchers have explored other microarchitectural components as the covert channel to transmit data during transient execution [2, 3]. However, it is considerably more challenging to implement these non-cache-based attacks because of the increased complexity of setting up the covert channels using other specific microarchitectural resources. In this paper, *we propose a novel timing side channel based on the timing of transient execution (ToTE), which can be conveniently used not only to build covert channels, but also to implement various side-channel attacks (SCA).*

Our research is inspired by the time difference of the Jcc (Jump if Condition Is Met) instructions during transient execution. Specifically, we observe that even when a Jcc instruction is not committed, its misprediction during transient execution leads to an additional pipeline stall. Moreover, the delay caused by this stall can be observed outside the transient execution phase through timing analysis of the Transient Execution Timing (TET). We refer to this as Whisper because it can be utilized to transmit invisible data fetched during transient execution.

We propose a list of novel side-channel attacks, including TET-Meltdown (MD), TET-Zombieload (ZBL), and TET-Spectre-V5-RSB (RSB), where the secrets fetched utilizing Meltdown [17], Zombieload [21] and Spectre-V5-RSB [14] are transmitted using Whisper as the covert channel. More importantly, we propose TET-KASLR to break the kernel address space layout randomization (KASLR), a popular mechanism to safeguard the kernel from attacks such as code reuse. TET-KASLR is so effective that it breaks KASLR even in meltdown-resistant CPUs with KPTI [11] enabled and FLARE [4] deployed, the latter is considered as the state-of-art defense. These attacks are successfully implemented on various CPUs[1] and they bring new perspectives to the security research of transient execution.

To investigate the root cause of Whisper, we develop a performance monitor unit (PMU) [6, 16] toolset for in-depth analysis of the microarchitectural behaviors when exploiting Whisper. Through automated testing, we find that numerous PMU events reflect the different behaviors under different time lengths of Whisper in Intel and AMD CPUs. By analyzing the events, we find that the branch misprediction in transient execution will lead to complex instruction stream conflicts, causing stalls in the frontend and the execution engine in the CPU microarchitecture. Besides, we find that states of the memory subsystem also have effects on Whisper. For example, the load from an unmapped virtual address causes multiple data translation lookaside buffer (DTLB) load misses that will decrease the disturbance from the memory subsystem to Whisper. We have tested that at least 3 types of Jcc instructions can be used, e.g. JE/JZ, JNE/JNZ, and JC instructions can lead to such a time difference. We believe that all the conditional jump instructions of x86 chips could be exploited and Whisper can be implemented on all of the tested Intel and AMD CPUs.

In summary, our contributions are as follows:

(1) We discover Whisper, a novel timing-based side channel on transient execution timing (TET), and develop a PMU toolset to investigate the root cause of this side channel.
(2) We propose several side-channel attacks based on Whisper that directly exploit the execution time of transient execution to disclose secret data or breach the deployed defenses.
(3) We successfully implemented the TET covert channel and side-channel attacks including Meltdown, Zombieload, and Spectre-RSB on a variety of x86 CPUs.

*Corresponding author

DAC '2024, June 23-27, 2024, San Francisco, CA, USA

[1]Responsible Disclosure: We have disclosed our work and results to the affected vendors. Intel acknowledged this on June 8, 2023.

(4) We propose TET-KASLR to break KASLR on multiple Intel CPUs with the protection of KPTI and/or FLARE.

2 BACKGROUND AND RELATED WORK

2.1 Microarchitecture Side-Channel Attacks

The modern CPU's microarchitecture (μarch) is an intricate system comprised of key components: the frontend, which is responsible for instruction fetch and decode; the out-of-order execution backend, which manages the out-of-order execution of instructions; and the memory subsystem, which temporarily stores frequently accessed data and instructions via cache. Despite its complexity, security vulnerabilities have been unearthed across most μarch components, including the Spectre attack [13] stemming from the branch predictor unit in the frontend, port contention side-channel attack [2] in the backend, and the cache side-channel attack [26] in the memory subsystem.

Kernel address space layout randomization (KASLR) enables address space randomization for the kernel image by randomizing where the kernel code is placed at boot time. It was deployed on Windows in 2007, macOS in 2012, and Linux in 2014 [4] to safeguard the kernel from code reuse attacks like return-oriented programming (ROP) [22]. Due to its popularity and critical role in modern operating systems security, KASLR has been a prime target for μarch side-channel attacks for over a decade. The first attack was unveiled in 2013 [12]. In response, Kernel page-table isolation (KPTI) [11] was proposed in 2017 as a universal mitigation by making better isolation between user space and kernel space memory. To addressing the residual exempted pages gaps in KPTI, FLARE (2020) [4] emerged as a state-of-the-art defense. However, new KASLR attacks continue to surface, such as Entrybleed (2023) [18] and AVX Timing (2023) [5].

Both the Entrybleed and AVX Timing attacks abuse the exempted pages of user-kernel isolation to break KASLR. The former exploits syscall and prefetch instruction, while the latter exploits AVX instruction. Instead of using any specific instructions, our work focuses on breaking KASLR with behavioral timing, instead of depending on specific instructions.

2.2 Transient Execution Attacks

Since Meltdown [17] and Spectre [13] were disclosed in 2018, transient execution attacks have become a hot topic in hardware security. The Meltdown [17] exploits transient execution in delayed fault handling to access secret data, while the Spectre [13] leverages the misprediction of branches predict unit (BPU), and Spectre-V5-RSB [14, 19] exploit the return stack buffer within BPU. The Zombieload [21] is a type of Microarchitectural Data Sampling (MDS) attack that utilizes the speculative buffer forwarding in the processor to extract temporary data. New transient execution attacks, such as Downfall (2023) [20], are still being reported.

In transient execution attack, the processes and their data are invisible and will be rolled back at the architectural level, necessitating the need for a side channel to transmit, receive, and decode the secret data. The most commonly used side channel in transient execution attacks is flush+reload [26]. Consequently, there has been some research focused on mitigating transient side effects to render the side-channel ineffective [25, 27] or detect attacks based on side-channel characteristics [9, 15]. As our TET side channel exploits stall in the core and is unrelated to the cache, the cache-based mitigation cannot address the TET side channel, which we have partially verified with our PMU toolset.

3 WHISPER: A TRANSIENT EXECUTION TIMING SIDE CHANNEL

3.1 Motivation

As the data within transient execution is invisible, a transient execution attack needs to find some side channel as a covert channel to transmit the data. Current research mainly focuses on probing the status of μarch components during or after the transient execution by timing the non-transient execution of some instructions, especially cache in the memory subsystem[13, 20]. Although these approaches have been successful, effective mitigation could be developed targeting directly the μarch component they utilize as the covert channel[15, 25].

To the best of our knowledge, this is the first attempt to investigate whether the execution time of transient execution (ToTE) can be leveraged to develop transient execution attacks. It is based on our observation that the system's transient state or behavior could influence ToTE and the impact can be observed through timing analysis outside of transient execution. There are two major advantages of our approach. First, the transient execution timing (TET) side channel exists and thus it reduces, if not completely avoids, the cost of constructing a covert channel with other μarch component. Second, since the TET side channel does not rely on any μarch components, it will be hard, if not impossible, for the existing mitigation mechanisms to detect such a side channel.

3.2 Discovering the TET Side Channel

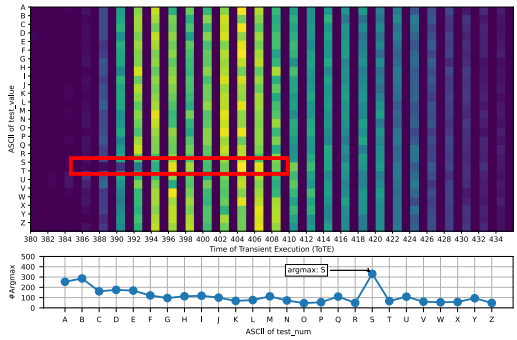
The execution time of an instruction depends on many factors including data, instruction, cache miss/hit, pipeline, etc. In our experiments, we observe that ToTE remains constant regardless of the data, instruction, and whether the transient access address is in the cache or not. However, we notice that data and control flow events may cause a timing difference of transient execution on different conditions (see the bottom of Figure 1b).

Figure 1 shows the gadget of the TET side channel on the Jcc (Jump if Condition Is Met) instruction and the result. The Figure 1a shows the gadget of TET. We get ToTE via recording the timestamps before and after this gadget through `RDTSC`. We use function `transient_begin` [4] either starts an Intel TSX transaction or sets up a signal handler to suppress exceptions. Line 3 is the key code of the TET Block, which we use to trigger Jcc during the transient execution caused by line 2. We iterate through the values of `test_value` from 0 to 255 in batches, and record the ToTE for each iteration. Executing multiple batches allows us to construct a frequency plot of ToTE, visualized in Figure 1b. In the highlighted region within the red box, it becomes non-trivial that the ToTE surpasses others when Jcc is triggered. This observation is further supported by the subsequent argmax plots. As a result, at the architecture level, we can observe the difference in ToTE caused by different jump conditions.

With this observation, we propose the TET side channel, which leaks data or control flow information during transient execution. We will analyze the root cause of the TET side channel in section 5

```
1  if (transient_begin())
2    *(char*)(0x0);    // Transient Block Start
3    if (test_value == 'S')  asm ("nop");
4    transient_end();  // Transient Block End
```

(a) The gadget that to be timing.



(b) Timing observation when condition is met in 'S'.

Figure 1: Gadget of TET and result.

and report that many instructions besides Jcc can lead to this side channel. We can leverage the TET side channel for transient execution attacks such as Meltdown, Zombieload, and Spectre-V5-RSB. We also discover an attack to break KASLR. Details of these attacks is provided below in section 4.

3.3 What's New in TET Side Channel?

As numerous covert channel and side-channel attacks (SCA) have been discovered, we compare the representative ones with our TET SCAs in Table 1 using categories expanded from Binoculars [28].

Table 1: Comparison of Side Channel Attacks.

Type	Probe Architecture		Transient-Only
	Stateful	Stateless	
Direct	Cache (e.g. Flush+Reload [26]) BPU [8]	Port Contention [2, 3] AVX [5], EntryBleed [18]	TET-MD, TET-ZBL TET-RSB
Indirect	TLB (e.g. TLBleed [10], AnC [24])	Binoculars [28]	TET-KASLR

In Table 1, *direct* refers to the attacks where results come directly from the victim's micro-operations. *Stateful* means that there is a persistent state change during the attack. *Stateful* SCAs, especially cache-based attacks, can be represented by a three-step model, state initial step, side effect step, and probing step [26]. It is known that the state change could be easily exploited and detected [15]. On the contrary, *stateless* channels been considered more difficult to exploit [28], and stateless SCAs will be difficult to detect or represent through a unified behavioral model. Most stateless SCAs are contention-based [2, 3, 28], where constructing the required contention synchronization is challenging.

Our proposed SCAs based on TET side channel are listed in the last column of Table 1. They are all stateless and hence robust against defenses. As we have discussed earlier, ToTE can be observed through architectural timing analysis, significantly reducing the implementation cost. TET SCAs are also transient-only, meaning that the information delivery during the attack does not depend on the changes in architectural state or the contention of microarchitectural components. So there is no need to change the architectural state within transient execution. As far as we know, TET side channel is the first transient-only covert channel for transient execution attacks. To summarize, our TET SCAs are stateless and transient-only, hence stealthy and easy to construct.

4 REAL-WORLD EXPLOITATION

4.1 Experiment Setup and Result

We summarize our experimentation in Table 2. All the tested machines running Ubuntu with Linux Kernel. Our experiments reveal that for 1k random bytes, the throughput of TET-CC could achieve 500B/s with an error rate of less than 5% at i7-7700, and the TET-MD can reach up to 50B/s with an error rate of less than 3% at i7-7700, and the TET-RSB can reach up to 21.5 KB/s with an error rate of less than 0.1% at i9-13900K. The TET-KASLR can break the KASLR in an average of 0.8829 s (n=3, μ = 0.0036) at i9-10980XE.

4.2 Threat Model and Assumption

We assume an unprivileged attacker could execute arbitrary instructions on the KASLR-enabled local machine. The attacker knows the CPU model and kernel image's constant offsets, the TLB can be evicted or invalid by other methods. For TET-MD and TET-ZBL, there is another process that runs on the same machine as a victim. We assume that the CPU has hardware vulnerabilities like Meltdown and Zombieload, but they have deployed state-of-art attack detection based on cache behavior. The attacker's target is to gain secret data within the victim's memory stealthily. In the case of breaking KASLR, the attacker's target is to find out the offset of KASLR to launch code reuse attacks.

4.3 TET for Transient Execution Attacks

4.3.1 TET-Meltdown (MD). We implement the TET-MD by using TET as the covert channel in the Meltdown attack. The attack is composed of two phases. In the first phase, we trigger transient execution and trigger Jcc instruction if the secret data obtained is equal to the test value. In the second phase, we record the execution time of the transient execution. Finally, by analyzing ToTE, we can obtain the value of secret data as we have shown in Figure 1. In our analysis method, we count the argmax of ToTE after traversing around the test value from 0 to 255. The argmax of the counting result is the secret value.

4.3.2 TET-Zombieload (ZBL). In the case of the original Zombieload attack, the aggressive forwarding within the microcode assisting faulting load during a page fault enables attackers to retrieve stale data from the victim from the line fill buffer. Then the attacker extracts this stale data out of the transient via flush+reload cache covert channel [21]. In TET-ZBL, the attacker extracts the stale data through TET via jump if the in-flight data is met with a test value, instead of the flush+reload. Contrary to TET-MD, it is interesting that the ToTE becomes shorter if the Jcc is triggered. Finally, by analyzing the time of ToTE like TET-MD, the value of stale secret can be obtained.

4.3.3 TET-Spectre-V5-RSB (RSB). We have preliminarily verified that through TET, we can extract the secret within the transient execution of Spectre-RSB. As we show in Listing 1, we make a function call on line 4 and jump to line 7. Line 5 will be pushed to return stack buffers (RSB) as the speculated return address. In lines

Table 2: Environment and experiments. ✓ means that we successfully achieved the attack, ✗ is the opposite. ? means not verify.

CPU	μ-arch	Microcode	Kernel	PMU-based Analysis	TET-CC	TET-MD	TET-ZBL	TET-RSB	TET-KASLR
Intel Core-i7-6700	Skylake	0xf0	4.15.0-213	TET-CC, TET-MD	✓	✓	✓	✓	✓
Intel Core-i7-7700	Kaby Lake	0x5e	5.4.0-150	TET-CC, TET-MD	✓	✓	✓	✓	✓
Intel Core-i9-10980XE	Comet Lake	0x5003303	5.15.0-72	TET-KASLR	✓	✗	✗	?	✓
Intel Core-i9-13900K	Raptor Lake	0x119	5.15.0-86	-	✓	✗	✗	✓	?
AMD Ryzen 5 5600G & 5900	Zen 3	0xA50000D	5.15.0-76	TET-CC	✓	✗	✗	?	✗

8 to 10, we modify the return address and flush the address where the return address is stored to increase the transient window length. When executing `retq`, the Spectre-RSB, which is caused by RSB incorrectly predicts, will issue a transient execution that returns to line 5 and performs Jcc operations that depend on the secret value. If a branch misprediction occurs in the transient execution caused by RSB, the entire transient time will be reduced like TET-ZBL. Through the final time analysis, we can obtain the value of the secret data that is obtained within TET-RSB.

```
1  for (test_value = 0; test_value <= 255; test_value++)
2      start_time = rdtsc();
3      lfence();       // ---- TET-RSB Block Start ----
4      asm ("call 1f;");
5      if (test_value == *(secret))    // Access secret and Jcc
6          asm ("nop;");
7      asm ("1: nop;"
8           "movabs $2f, %%rax;"
9           "mov %%eax, (%%rsp);"
10          "clflush (%%rsp);"
11          "retq;"                    // RSB misprediction
12          "2:;" ::: "rax");
13     lfence();         // ==== TET-RSB Block End ====
14     spend_time = rdtsc() - start_time;
15     if (max_time < spend_time)
16         max_time = spend_time, argmax = test_value;
```

Listing 1: Pseudocode for TET-RSB

4.4 Covert Channel for SMT

As the exception would cause pipeline flush, we can abuse it to build a covert channel between two simultaneous multithreading (SMT). The sender (Trojan) thread triggers page fault and suppresses it through the signal handler to send '1', otherwise, send 0, while the receiver (spy) just records the time of the nop loop. By distinguishing the nop loop time, the spy can analyze and obtain the data sent by the Trojan. Our prototype verification speed was 1B/s with an error rate lower than 5% in Core i7-7700. Using the evaluate tools from SecSMT [23], the preliminary throughput could achieve 268 KB/s though with a 28% error rate. We leave speed up with high accuracy and further exploit to future work.

4.5 TET-KASLR attack: Breaking KASLR

As transient execution is caused by triggering page faults of accessing illegal addresses, we explore how different address types such as illegal mapped or unmapped addresses influence the ToTE. We notice that if the address that triggers the exception is mapped, the ToTE will be shorter than unmapped, and the pipeline stall caused by Jcc will be absent. For the root cause of this evidence, as we referred to previous research [12] and the Table 3, we doubt that Intel's CPUs will trigger the loading of TLB entries for mapped addresses, even for illegal access without permission. But access to an unmapped address will not be able to trigger the real loading of TLB entries from the cache which extends the page walk and reduces the interference to observe pipeline stall.

Exploiting TET's ability of detecting mapped addresses, and the fixed mapped location of the Linux kernel image is within the interval `0xffffffff80000000-0xffffffffc0000000` with 4-KiB alignment [5], we find that the protection of KASLR can be compromised by mapping detection within this range. The attacker can flush or evict [12] the TLB and then conduct a TET-KASLR probe at potential addresses' outset until discovering the first mapped address, which marking the initiation of the kernel image address. Consequently, we successfully break KASLR in Intel i7-6700, i7-7700, and i9-10980XE.

```
1  mfence();
2  asm ("mov %0, %%rbx; movq (%1), %%rax; sub (%2), %%rbx;"
3      :: "r"(test_num), "c"(invalid_addr), "r"(&secret):);
4  asm ("jz 1f; jmp 2f; 1: nop;");
5  printf("you can not see me");
6  asm ("2: nop; mfence;");
```

Listing 2: Pseudocode transient block for TET-KASLR

Breaking KASLR with KPTI enabled. Since KPTI still retains the necessary kernel mappings at user addresses, we can use this remnant trampoline at fixed offset `0xe00000` to break KASLR after enabling KPTI. In i9-10980XE, we traverse the 512 possible offsets of KASLR and successfully detect the location of the KPTI trampoline within 1s. We further use the TET-KASLR to bypass FLARE [4], a state-of-the-art defense against currently known KASLR breaks.

Breaking KASLR in Virtualization Environments. Our empirical shows that using TET-KASLR could even break KASLR successfully in the Docker environment (Docker version 24.0.1, build 6802122, runc runtime) in i9-10980XE.

5 ANALYSIS OF WHISPER

5.1 Method

We use PMU to analyze the TET because the PMU has some events that can reflect the behavior of the front end, execution engine, and storage subsystem. Since manual analysis and testing hundreds of PMU events are a daunting task and challenge, we designed an automated testing toolset for better and more convenient analysis.

We divided our analysis using this toolset into three stages, as shown in Figure 2, preparation stage, online collection stage, and offline analysis stage. In the first stage of collecting PMU events that can be used for analysis, we used the data provided within Intel's Perfmon and Linux Perf tools to obtain all possible PMU event records. In the following stage, we can obtain a large amount of raw data that can be used for analysis. This raw data can be filtered out by simple differential methods to filter out the irrelevant parts.

Based on differential filtering of PMU experimental results, we obtained PMU events related to transient timing differences and produced an in-depth analysis. We selected the critical events and listed their counter value in Table 3.

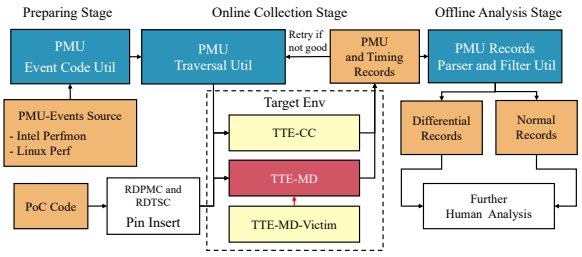


Figure 2: Analysis flow using PMU toolset.

5.2 Microarchitectural Level Analysis for TET

We performed experimental evaluations on various processors (Table 2) and used the result (Table 3) to address the following research questions: (**RQ1**) How does the frontend affect the ToTE? (**RQ2**) How does the backend affect the ToTE? (**RQ3**) How does the memory subsystem affect the ToTE?

5.2.1 The True Negatives Result. Firstly, through the result of CYCLE_ACTIVITY. CYCLES_MEM_ANY event from Table 3, we ruled out the impact of the memory-related stall.

5.2.2 Analysis of Frontend. We observed the counter for two undocumented events in skylake chips named BR_MISP_EXEC.INDIRECT and BR_MISP_EXEC.ALL_BRANCHES will increase when Jcc instruction is triggered. Also, the cycles that no μops executed will be increased when Jcc is triggered. So we suspect that it is the stall side-effect from misprediction in frontend within transient execution that leads to the timing difference, shown in Figure 3. In addition, we also observed that the instruction from the decode stream buffer (DSB, a.k.a. micro-operation cache) is decreased, and more μops will be delivered to backend from micro-instruction translation engine (MITE, a.k.a. legacy frontend). This gives **Answer to RQ1:** Resteer of BPU misprediction causes transient stall.

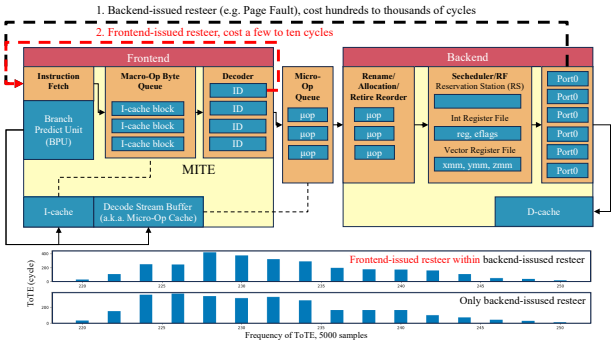


Figure 3: Fronted-issued resteer within transient execution.

5.2.3 Analysis of Pipeline and Backend. Summarizing the PMU events about the pipeline and backend of Table 3, we can see that resource-related stalls occur in the pipeline, the number of microcode instructions sent by the frontend to the backend decreases, and the backend's reserved stack waits for the frontend to send μops. Again, our suspicions are reflected. Hence, we have **Answer to RQ2:** The resource related stall of the pipeline.

Table 3: Key performance monitor counter values.

CPU & Scene	Event Name	Jcc not Trigger	Jcc Trigger
Core i7-6700 TTE-CC	BR_MISP_EXEC.INDIRECT	0	1
	BR_MISP_EXEC.ALL_BRANCHES	0	2
	RESOURCE_STALLS.ANY	15	21
Core i7-7700 TTE-CC	**BR_MISP_EXEC.INDIRECT**	0	1
	BR_MISP_EXEC.ALL_BRANCHES	0	2
Core i7-7700 TTE-MD	**IDQ.DSB_UOPS**	119	115
	IDQ.MS_DSB_CYCLES	33	26
	IDQ.DSB_CYCLES_OK	54	43
	IDQ.DSB_CYCLES_ANY	76	60
	IDQ.MS_MITE_UOPS	77	97
	IDQ.ALL_MITE_CYCLES_ANY_UOPS	35	45
	IDQ.MS_UOPS	228	208
	UOPS_EXECUTED.CORE_CYCLES_NONE	110	116
	RESOURCE_STALLS.ANY	15	21
	CYCLE_ACTIVITY.STALLS_TOTAL	320	331
	UOPS_EXECUTED.STALL_CYCLES	325	332
	CYCLE_ACTIVITY.CYCLES_MEM_ANY	142	141
	INT_MISC.RECOVERY_CYCLES_ANY	24	29
	INT_MISC.CLEAR_RESTEER_CYCLES	27	39
	UOPS_ISSUED.ANY	334	319
	UOPS_ISSUED.STALL_CYCLES	394	404
	RS_EVENTS.EMPTY_CYCLES	202	218
Ryzen 5 5600G TTE-CC	bp_l1_btb_correct	493	511
	bp_l1_tlb_fetch_hit	914	938
	de_dis_uop_queue_empty_di0	182	195
	de_dis_dispatch_token_stalls2.retire_token_stall	4	84
	ic_fw32	661	690
Core i7-6700 Transient Execution Flow	UOPS_ISSUED.ANY	684	603
	INT_MISC.RECOVERY_CYCLES	19	15
	ICACHE_16B.IFDATA_STALL	2	0

CPU	Event Name	unmapped	mapped
Core i9-10980XE TTE-KASLR	DTLB_LOAD_MISSES.MISS_CAUSES_A_WALK	2	0
	DTLB_LOAD_MISSES.WALK_ACTIVE	62	0
	ITLB_MISSES.WALK_ACTIVE	19	0

5.2.4 Analysis of Memory Subsystem. We observed that if the virtual address target of illegal access is mapped to a physical address, it impacts the timing of the transient execution (ToTE). Specifically, it will make ToTE longer. Furthermore, we observed that when ToTE becomes shorter and the timing difference from the misprediction of Jcc would be unobservable, some related PMU events change, as listed in Table 3. This suggests **Answer to RQ3:** The TLB missing could extend the ToTE.

5.2.5 Analysis of Transient Execution Flow. We study branch reachability in transient states and show the control flow graph of the experiment in Figure 4. For UOPS_ISSUED.ANY. We analyze that the Jcc not trigger path will encounter a fence, which hinders the issuance of subsequent μops. In contrast, the trigger path does not meet a fence, resulting in the issuance of an increasing number of μops. This event confirms the existence of the trigger path ③. If the number of nop instructions preceding the mfence is increased, such that the not trigger path does not encounter the mfence before the rollback, the opposite result is obtained, with fewer μops being issued in the trigger path. This could be attributed to the branch rollback occurring in the trigger path, causing a certain delay in the issuance process. For INT_MISC.RECOVERY_CYCLES, we analyze that in the trigger path, the presence of stage ② causes a temporary pause in the dispatch unit, suggesting the occurrence of additional recovery stall within the transient window.

6 SECURITY DISCUSSION

6.1 Deficiency of existing defenses

TET shows that it is not enough to mitigate microarchitecture vulnerabilities via detecting or blocking covert and side channels

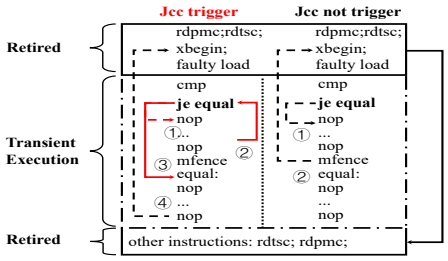


Figure 4: Control flow graph for the Transient Execution.

that rely on the specific microarchitectural components as observing the execution time of transient execution does not need these components. We demonstrate that KPTI and FLARE are no longer sufficient to protect KASLR. Nor is the method of replacing AVX instructions [5] as the attacker can exploit the TLB's vulnerable behavior in completely different ways.

6.2 Software-based Mitigation

For TET-MD and TET-ZBL, the KPTI and the microcode updates released by Intel are efficient mitigation. To mitigate TET-KASLR, one can use FGKASLR [1], which reorders individual kernel functions such that even when the offset of the kernel address is exposed, the attacker is still unable to pinpoint the specific functions based on their relative addresses. However, such mitigation comes with high performance overhead.

6.3 Hardware-based Mitigation

Our findings indicate that TLB entries should only be created if the access permission check is passed. Otherwise, it may introduce vulnerable side effects to the operating system. While the implementation of a USER/KERNEL-separated TLB or the security TLB [7], has the potential to fully mitigate TET-KASLR and other TLB-related vulnerabilities, it might be expensive and less practical to replace the implementation of current hardware.

7 CONCLUSION

We report a novel timing side channel based on the execution time of transient execution. Using such side channel, we have successfully implemented many known side-channel attacks (SCA) including Meltdown, Zombieload and Spectre-RSB. More importantly, we show that KASLR under the state-of-the-art protection of KPTI and FLARE can also be broken with the proposed transient execution timing (TET) side channel. We verify on commercial CPUs that the proposed TET SCAs are stealthy and easy to construct.

ACKNOWLEDGMENT

The authors of BUPT were supported in part by the National Key Research and Development Program of China (Grant No. 2023YFB4403000), Beijing Natural Science Foundation (Grant No. 4242026), and the Fundamental Research Funds for the Central Universities (Grant No. 2023RC71). The authors of Tsinghua University were supported in part by the National Natural Science Foundation of China (Grant No. 62072263) and Tsinghua University Initiative Scientific Research Program.

REFERENCES

[1] Kristen Carlson Accardi. 2020. Function Granular KASLR. https://lwn.net/Articles/824307/.
[2] Alejandro Cabrera Aldaya, Billy Bob Brumley, Sohaib ul Hassan, Cesar Pereida García, and Nicola Tuveri. 2019. Port contention for fun and profit. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 870–887.
[3] Atri Bhattacharyya, Alexandra Sandulescu, Matthias Neugschwandtner, Alessandro Sorniotti, Babak Falsafi, Mathias Payer, and Anil Kurmus. 2019. Smotherspectre: exploiting speculative execution through port contention. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 785–800.
[4] Claudio Canella, Michael Schwarz, Martin Haubenwallner, Martin Schwarzl, and Daniel Gruss. 2020. KASLR: Break It, Fix It, Repeat. In *15th ACM Asia Conference on Computer and Communications Security (ASIACCS)*.
[5] Hyunwoo Choi, Suryeon Kim, and Seungwon Shin. 2023. AVX Timing Side-Channel Attacks against Address Space Layout Randomization. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[6] Intel Corporation. 2022. Intel® 64 and IA-32 Architectures Software Developer's Manual.
[7] Shuwen Deng, Wenjie Xiong, and Jakub Szefer. 2019. Secure tlbs. In *Proceedings of the 46th International Symposium on Computer Architecture*. 346–359.
[8] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices* 53, 2 (2018), 693–707.
[9] Xaver Fabian, Marco Patrignani, and Marco Guarnieri. 2022. Automatic Detection of Speculative Execution Combinations. In *CCS*.
[10] Ben Gras, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2018. Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks. In *27th USENIX Security Symposium (USENIX Security 18)*. 955–972.
[11] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. 2017. Kaslr is dead: long live kaslr. In *Engineering Secure Software and Systems: 9th International Symposium, ESSoS 2017, Bonn, Germany, July 3-5, 2017, Proceedings 9*. Springer, 161–176.
[12] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical Timing Side Channel Attacks against Kernel Space ASLR. In *2013 IEEE Symposium on Security and Privacy*. 191–205. https://doi.org/10.1109/SP.2013.23
[13] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre Attacks: Exploiting Speculative Execution. In *S&P*.
[14] Esmaeil Mohammadian Koruyeh, Khaled N Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre Returns! Speculation Attacks using the Return Stack Buffer. In *WOOT*.
[15] Congmiao Li and Jean-Luc Gaudiot. 2022. Detecting Spectre Attacks Using Hardware Performance Counters. *IEEE Trans. Comput.* (2022).
[16] AMD Limited. 2023. AMD64 Architecture Programmer's Manual, Volumes 1-5.
[17] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading Kernel Memory from User Space. In *Usenix Security*.
[18] William Liu, Joseph Ravichandran, and Mengjia Yan. 2023. EntryBleed: A Universal KASLR Bypass against KPTI on Linux. In *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*. 10–18.
[19] Giorgi Maisuradze and Christian Rossow. 2018. ret2spec: Speculative Execution Using Return Stack Buffers. In *CCS*.
[20] Daniel Moghimi. 2023. Downfall: Exploiting Speculative Data Gathering. In *32th USENIX Security Symposium (USENIX Security 2023)*.
[21] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad : Cross-Privilege-Boundary Data Sampling. In *CCS*.
[22] Hovav Shacham. 2007. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). In *14th ACM Conference on Computer and Communications Security (CCS)*.
[23] Mohammadkazem Taram, Xida Ren, Ashish Venkat, and Dean Tullsen. 2022. SecSMT: Securing SMT Processors against Contention-Based Covert Channels. In *31st USENIX Security Symposium (USENIX Security 22)*. 3165–3182.
[24] Stephan Van Schaik, Kaveh Razavi, Ben Gras, Herbert Bos, and Cristiano Giuffrida. 2017. RevAnC: A framework for reverse engineering hardware page table caches. In *Proceedings of the 10th European Workshop on Systems Security*. 1–6.
[25] Mengjia Yan, Jiho Choi, Dimitrios Skarlatos, Adam Morrison, Christopher Fletcher, and Josep Torrellas. 2018. Invisispec: Making speculative execution invisible in the cache hierarchy. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 428–441.
[26] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. 719–732.
[27] Jiyong Yu, Mengjia Yan, Artem Khyzha, Adam Morrison, Josep Torrellas, and Christopher W Fletcher. 2019. Speculative taint tracking (stt) a comprehensive protection for speculatively accessed data. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 954–968.
[28] Zirui Neil Zhao, Adam Morrison, Christopher W Fletcher, and Josep Torrellas. 2022. Binoculars: Contention-Based Side-Channel Attacks Exploiting the Page Walker. In *31st USENIX Security Symposium (USENIX Security 22)*. 699–716.