TF-Timer: Mitigating Cache Side-Channel Attacks in Cloud through a Targeted Fuzzy Timer

Mingyu Wang*†, Shijie Jia*, Fangyu Zheng‡, Yuan Ma*, Jingqiang Lin§, Lingjia Meng*† and Ziqiang Ma¶
* Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China
† School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
‡ School of Cryptology, University of Chinese Academy of Sciences, Beijing, China
§School of Cyber Security, University of Science and Technology of China, Hefei, China
¶School of Information Engineering, Ningxia University, Yinchuan, China

2024 IEEE Wireless Communications and Networking Conference (WCNC) | 979-8-3503-0358-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/WCNC57260.2024.10571330

***Abstract*—Cache side-channel attacks pose a significant threat to the data security of multi-tenant public clouds. However, currently proposed defenses either lack transparency (requiring user involvement) or incur a significant performance penalty. This paper is motivated by our insightful observation for the behavior of cache side-channel attackers who employ `rdtsc/rdtscp` instructions for timing purposes. We have discerned a behavior pattern that enables comprehensive identification of potential attackers. Building upon this observation, we introduce TF-timer, which operates on the core principle of inspecting cache side-channel attacks using the pre-identified behavior pattern while obscuring the return values of `rdtsc/rdtscp` instructions. Our proposed technique preserves the properties of `rdtsc/rdtscp`, only blurring the attacker's timing to minimize the impact on other applications. We have implemented the prototype of TF-timer at the hypervisor layer. It is completely transparent to users and requires no hardware modifications. Our evaluation results demonstrate that TF-timer efficiently and precisely mitigates cache side-channel attacks that exploit `rdtsc/rdtscp` for timing, with performance penalties within 1%.**

***Index Terms*—Cache side-channel attacks, Intel VT, Fuzzy timer**

I. Introduction

In recent decades, cloud computing has seen widespread adoption in optimizing the efficiency of computer hardware resource utilization. However, its security has become vulnerable to exploitation through various attacks [1], [2]. One of the most noteworthy concerns in the public cloud with multiple tenants is cache side-channel attack [3], which commonly utilizes `rdtsc/rdtscp` instructions to measure the latency of microarchitectural events (e.g., memory access, flush and prefetch), and deduces the victim's sensitive operations based on the latency [3]. Various cache side-channel attack variants have been introduced in the cloud, such as FLUSH+RELOAD [2] and FLUSH+FLUSH [4]. These attacks can be used to steal sensitive values (e.g., cryptographic keys [5]), spy on user privacy information [6], retrieve TLS (Transport Layer Security) messages [7] and hijack accounts on web-servers within a PaaS cloud [8].

This work was supported by National Natural Science Foundation of China (No.62272457), National Key Research and Development Program of China (No.2022YFB3103301), and Key RD plan of Ningxia Hui Autonomous Region, China (No.2021BEB04047).

Corresponding author: Shijie Jia (Email: jiashijie@iie.ac.cn).

To defend against cache side-channel attacks, various defense mechanisms have been proposed to date. Among them, a commonly employed lightweight approach focuses on obfuscating the timer utilized by cache-side channel attackers [3]. The most common timing method used in cache side-channel attacks is taking advantage of the `rdtsc/rdtscp` instruction to read the Time-Stamp Counter (TSC) on x86 processors [9]. Therefore, decreasing the fidelity of TSC has been advocated in previous defenses [10], [3]. While many efforts have been devoted to decreasing the fidelity of TSC to defend against cache side-channel attacks, little attention has been paid to achieving a balance between transparency, accuracy, and efficiency. Previous efforts either require users to determine when and how to obscure the TSC [3], [11], or introduce hardware modifications [9], [10]. Furthermore, some methods obscure the return values of all `rdtsc/rdtscp` instructions indiscriminately [3], [12], thus affecting the benign processes.

In this paper, we focus on addressing the transparency and accuracy issues of previous defenses, meanwhile achieving efficiency. Specifically, we tackle the aforementioned limitations and propose TF-Timer, a targeted fuzzing timer to identify and mitigate cache side-channel attacks that use `rdtsc/rdtscp` instructions for timing. TF-Timer consists of two parts, an inspector and a fuzzer. In order to identify potential cache side-channel attackers, we observe and summarize the common characteristics of the behavior patterns of all the existing cache side-channel attacks. Taking advantage of the common behavior patterns, the inspector captures `rdtsc/rdtscp` instructions in guest VMs and identifies potential cache side-channel attackers by checking the behavior patterns of the processes. The fuzzer only obfuscates the return value of the identified attacker's `rdtsc/rdtscp` instructions by subtracting a random number. TF-Timer achieves three goals simultaneously: (1) **Transparency**. TF-Timer is implemented at the hypervisor layer, which requires no modification of user applications or hardware, thus it is completely transparent to users and easy to deploy. (2) **Accuracy**. With the help of the inspector, TF-Timer only obfuscates the timer of the potential attack processes rather than all the processes. (3) **Efficiency**. Utilizing Intel hardware-assisted virtualization technology, which is widely supported on most processors,

TF-Timer only introduces a small modification in the Linux kernel and a small amount of system performance overhead.

In summary, the contributions of this paper are as follows:

- We investigate the common characteristics of cache side-channel attacks comprehensively and propose a TMT (Timing Microarchitectural-event Timing) behavior pattern to identify potential cache side-channel attackers that use `rdtsc/rdtscp` instructions for timing.
- We propose a targeted fuzzy timer, TF-Timer, which is implemented at the hypervisor layer and can mitigate cache side-channel attacks that rely on `rdtsc/rdtscp` instructions for timing microarchitectural events. Taking advantage of Intel hardware-assisted virtualization technology, TF-Timer only obfuscates the return value of the attackers' `rdtsc/rdtscp` instructions efficiently to minimize the impact on other applications.
- We implement a prototype of TF-Timer and conduct detailed evaluations of its security and performance. The evaluation results demonstrate that TF-Timer can alleviate mainstream cache side-channel attacks within 1% performance penalties.

II. Background

In this section, we provide a concise introduction to the related foundational concepts, encompassing cache side-channel attacks and Intel hardware-assisted virtualization.

A. Cache Side-Channel Attacks

Cache side-channel attacks commonly exploit the time discrepancy between a cache hit and a cache miss to infer the sensitive information (e.g., cryptographic keys) of the victim. Depending on the necessity of timing, cache side-channel attacks can be categorized into timing-dependent attacks and timing-independent attacks.

1) Timing-Dependent Attacks: According to the microarchitectural events the attacker utilizes, timing-dependent attacks can be further classified into memory-accessing-dependent attacks, flushing-dependent attacks, and prefetching-dependent attacks.

PRIME+PROBE [13], [14] is a typical memory-accessing-dependent attack including three phases: PRIME fills the target cache set with the eviction set; then the attacker waits for victim operations, at which point the victim may access the target cache set or not; finally PROBE accesses the target cache set and measures the probe latency to determine whether the victim has accessed the target cache set. PRIME+SCOPE [15] and RELOAD+REFRESH [16] are variants of PRIME+PROBE, involving cache access and latency measurements. PRIME+SCOPE targets a specific cache line, while RELOAD+REFRESH offers better concealment as it does not evict target data from the LLC (Last Level Cache).

FLUSH+RELOAD [2] is another typical memory-accessing-dependent attack, which takes advantage of memory sharing and LLC sharing across multiple cores. The attacker first flushes the target cache line using `clflush`; then waits for the victim's cryptographic operations to update the cache state; finally reloads the evicted data, measuring reload time to determine if the victim accessed the target cache line. PREFECH+RELOAD [17] is a variant of FLUSH+RELOAD that replaces `clflush` with `prefetchw` for the attack and also requires latency measurements to detect victim activity.

FLUSH+FLUSH [4] is a typical flushing-dependent attack, whose principle is similar to FLUSH+RELOAD. Their difference is that FLUSH+FLUSH uses the `clflush` instruction to evict the target memory block in the third step instead of reloading it. The attacker distinguishes a cache hit or a cache miss based on the execution time of the `clflush` instruction.

As a typical prefetch-dependent attack, instead of measuring the latency of accessing the target address, PREFETCH +PREFETCH [17] measures the latency of prefetching the target address to decide if the victim has accessed the said address.

2) Timing-Independent Attacks: PRIME+ABORT [18] utilizes Intel Transactional Synchronization Extensions (TSX) to monitor an abort that is related to a target memory region access instead of measuring the time of accessing the eviction set.

As far as we know, timing-dependent attacks currently represent the prevailing technique for cache side-channel attacks. In order to determine whether there is a cache hit or not, timing-dependent attacks commonly use `rdtsc/rdtscp` instructions [10] to measure the latency of accessing the target address or executing the flush, prefetch instructions.

B. Intel Hardware-assisted Virtualization

To enhance the x86 processor's support for virtualization, Intel introduced Intel Virtualization Technology (Intel VT), a hardware-assisted virtualization technology based on the x86 architecture. By introducing new instructions and operating modes, Virtual Machine Monitor (VMM) and guest software can run in Virtual Machine Extension (VMX) root operation and VMX non-root operation, respectively.

In non-root operation, a VM exits to the VMM when certain instructions are executed within the VM. The VMM handles the exit based on its exit reason, then returns to the VM with a VM entry. VMX facilitates CPU virtualization through the Virtual Machine Control Structure (VMCS), which manages register contents and control information. VMCS is updated during VM entry and exit. Specific instructions can be utilized to configure exit conditions by modifying the VMCS.

III. Design

In this section, we first describe our threat model, and then we investigate and summarize the common behavior pattern of the timing-dependent attacks. Finally, we introduce TF-timer in detail.

A. Threat Model

In this work, we focus on timing-dependent attacks in virtualized scenarios, where the attacker and the victim can run not only on the same VM but also on different VMs or on

different processors. We assume the attacker has no privileges, and the host and VMM are trusted. The host OS is running on an x86 processor. For other architectures (e.g., ARM), there are no registers and assembly instructions that directly correspond to TSC and `rdtsc/rdtscp` respectively [19]. Therefore, we do not consider these architectures in this paper and leave them for future work. In such a scenario, the attacker can continuously measure the latency of accessing, flushing, or prefetching sensitive cache lines by using `rdtsc/rdtscp` instructions and infer the victim's access pattern to sensitive cache lines through the latency, thereby stealing the victim's sensitive information.

B. Common Characteristics of Attack Behaviors

To realize the identification of timing-dependent attacks utilizing `rdtsc/rdtscp` instructions, we need to first conclude their common characteristics. We will call `rdtsc/rdtscp` instructions "timing instructions" in the rest of this paper.

In the existing timing-dependent attacks [2], [16], [15], a common code snippet in Listing 1 is utilized to discern cache hits from cache misses through latency measurement. This code snippet employs a specific timing instruction marked as T_1 (line 3). The attacker then triggers a microarchitectural event, such as a memory access operation (line 5). Subsequently, another timing instruction marked as T_2 is executed (line 6), and the values obtained from T_1 and T_2 are denoted as TV_1 and TV_2, respectively. The latency, calculated as $TV_2 - TV_1$, represents the execution time of the respective microarchitectural event. We name the above behavior pattern of a timing-dependent attack as **TMT pattern**, namely **Timing (T_1) Microarchitectural-event Timing (T_2) pattern**.

Listing 1: A classic core code from [2] to measure cache access latency.

```
1 mfence
2 lfence
3 rdtscp
4 mov %eax,%esi
5 mov (%r9),%eax
6 rdtscp
7 sub %esi,%eax
```

According to our investigation to the cache side-channel attacks introduced in Section II-A, we can see that all the timing-dependent attacks match the TMT pattern when measuring microarchitectural events. We refer to these attacks conforming to the TMT pattern as TMT attacks.

C. Architecture

Based on the behavior pattern of TMT attackers, we design an effective mitigation method named TF-Timer to defend against TMT attacks. The architecture of TF-Timer is shown in Fig. 1. TF-Timer mainly consists of two modules: an inspector and a fuzzer. The inspector captures the timing instructions executed by a process within a VM and checks the behavior pattern of the process. If there is a process whose behavior pattern conforms to the TMT pattern, the fuzzer will obfuscate the least significant few bits for mitigating TMT attacks. Next, we introduce the details of these two components respectively.

1) Inspector: The inspector captures the timing instructions executed within the guest VM through VM exits. Then, the inspector retrieves the upcoming instructions to determine whether their behavior pattern conforms to the TMT pattern.

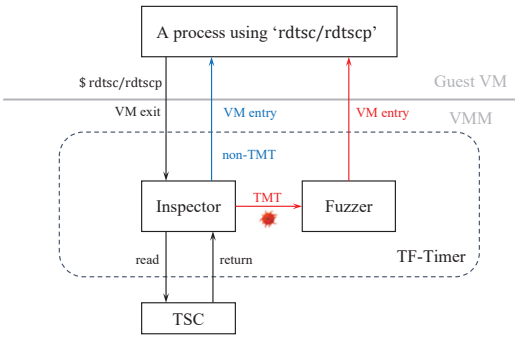


Fig. 1: TF-Timer Architecture.

As shown in Fig. 1, if a process in a guest VM executes `rdtsc/rdtscp` instructions, an *RDTSC/RDTSCP* exit event occurs. If TF-Timer captures an *RDTSC/RDTSCP* exit, the inspector gets the *guest_TSC* from TSC normally. Then, the inspector checks whether the current process is a potential TMT attacker.

Specifically, the inspector pre-reads S bytes instructions (a candidate code snippet) that will be executed by the current process. We refer to S as *code scanning boundary* that can be flexibly adjusted. By analyzing the candidate code snippet which will manifest as specific program behaviors, the inspector checks whether the behaviors conform to the TMT pattern. If so, we consider there is a potential attacker. As indicated by the red line in Fig. 1, it is necessary for the fuzzer to obfuscate the return value of the timing instruction of the current process (i.e., a potential TMT attacker), then the CPU returns to the guest VM from VMM (i.e., VM entry). If not, as indicated by the blue line in Fig. 1, the inspector returns the *guest_TSC* without fuzzing.

2) Fuzzer: The fuzzer obfuscates the return value of a T_1 in a potential TMT attacker to mitigate TMT attacks. In detail, TF-Timer obfuscates the least N significant bits of the return value of the T_1. N is a *fuzzy factor* that can be flexibly adjusted.

First, the fuzzer should obfuscate the TSC silently, making the obfuscation imperceptible to TMT attackers. To achieve this, TF-Timer generates an N-bit random number and subtracts the random number from the *guest_TSC*.

Second, to guarantee the normal behaviors of the benign processes in the system that need to use `rdtsc/rdtscp`, the properties of `rdtsc/rdtscp` should be maintained as much as possible. Specifically, There are four properties that should be taken into account [10]. ① **Incremental.** The return value of `rdtsc/rdtscp` should be strictly increasing. ② **Unpredictable.** The least significant bits of the return value of the `rdtsc/rdtscp` instruction are sometimes to be used to gather entropy. Thus, they should be unpredictable. ③ **Differential.** The cycle difference between two calls of the timing instruction should be reflected precisely as much as possible. ④ **Accurate.** The timing instruction should accurately reflect

the number of cycles the processor has been running. We will describe how to maintain these properties and how to obfuscate the value returned by `rdtsc/rdtscp` instructions silently in Section IV-C.

IV. Implementation

TF-timer is implemented at the VMM layer. We implement a prototype of TF-timer as an extension component of Linux kernel 4.15. In this section, we give the implementation details.

A. Matching the behavior pattern of a process

The detailed processing procedure of TF-Timer is shown in Fig. 2. To capture the timing instructions from guest VMs, TF-Timer sets the `RDTSC_EXITING` flag as 1 once a vCPU is scheduled. Thus, if a timing instruction is executed in a VM, the VM exits to VMM. We register new handlers in the hypervisor layer for the RDTSC/RDTSCP exit event.

As presented in Fig. 2, if an RDTSC/RDTSCP exit event occurs (Step 1), TF-Timer first computes *guest_TSC* from physical TSC. Afterwards, TF-Timer uses the `vmx_get_cpl()` function to obtain the current privilege level (CPL) of the current vCPU and checks whether the CPL is Ring 0, Ring 1, or Ring 2 (Step 2). If so, TF-Timer directly imports the *guest_TSC* into EDX:EAX and returns to the VM to continue executing subsequent instructions as usual (Step 3-1), thus filtering out the timing instructions coming from the kernel. Otherwise, if the timing instruction is executed in Ring 3, TF-Timer pre-reads S bytes instructions that will be executed after the current timing instruction (Step 3-2). Specifically, the inspector reads the instruction pointer (RIP) of the current VM through `vmcs_readl(GUEST_RIP)`, and pre-reads S bytes after the current timing instruction through `kvm_read_guest_virt()` to determine whether there is a microarchitectural event (Step 4).

TF-Timer identifies three events (i.e., memory access, flush, prefetch) through corresponding instructions, such as `clflush`, `prefetchnta`, `prefetchw`, etc. If there are no such instructions, TF-Timer returns the *guest_TSC* directly (Step 5-1). If there is such an instruction, TF-Timer checks the rest of the S bytes to decide whether there is another timing instruction. If there is another timing instruction, as the red line in Fig. 2 (Step 6) shows, TF-Timer disturbs the least significant N bits of *guest_TSC* and returns to the VM.

B. Selecting the code scanning boundary

The code scanning boundary (S) should be meticulously adjusted. If S is too small, the attacker could bypass the inspector by modifying their attack behavior. If S is too large, it might result in a high false positive rate. We have determined an appropriate value for S based on a series of test experiments.

Configurations of experiments. In our tests, we imitate a victim who keeps calculating ECDSA signatures by using the algorithm in OpenSSL 1.1.0h. The attacker deduces the digits of the ephemeral key k through a typical timing-dependent attack, FLUSH+RELOAD. This attack contains the code snippet depicted in Lisiting 1. In order to analyze and

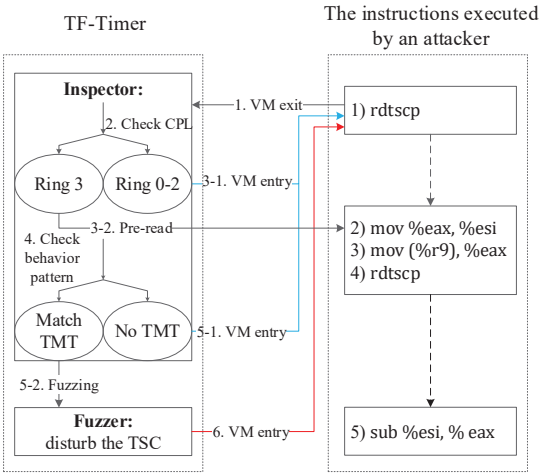
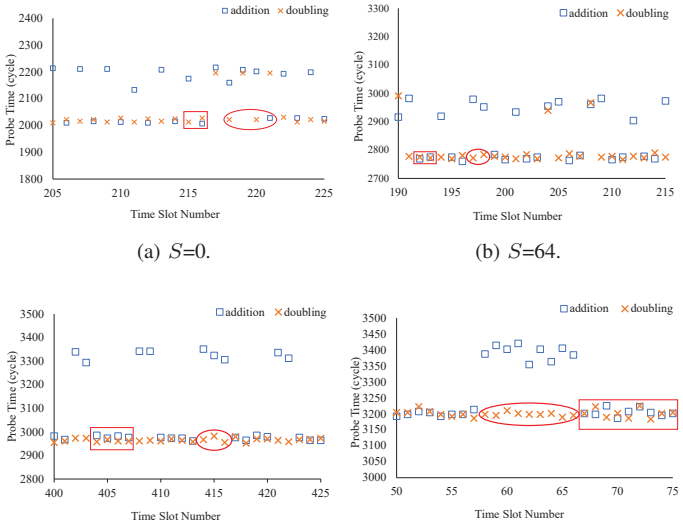


Fig. 2: The detailed processing of TF-Timer.

observe the captured attack results more clearly, we manually fix the ephemeral key k in each ECDSA signature calculation to '010101.....' instead of a random number. Specifically, if the digit of k is '0', the attacker can observe only a point doubling ('D'); if the digit of k is '1', there should be a point doubling and a point addition ('A') observed by the attacker. Therefore, the correct sequence of operations that an attacker observes should be *'DDADDA.....'*.

Testing the effects of varying S. To find an appropriate S, we test different values of S for TF-Timer. To simulate the potential evasion tactics employed by an attacker, we insert S `nop` instructions after T_1 for a FLUSH+RELOAD attack and gather the cycle difference calculated by $TV_2 - TV_1$.

Our tests start from $S = 0$. The result is shown in Fig. 3a, where we can clearly identify the sequence of 'D' and 'A' corresponding to each digit of k. The red box and circle in Fig. 3a both indicate that the two digits of k are '01'. Finally, the attacker can correctly infer that the digits of k are '010101.....'.



(a) S=0. (b) S=64.

(c) S=128. (d) S=192.

Fig. 3: The results of a FLUSH+RELOAD attack with different values of S.

When the value of S is set as 64, 128, and 196, the attacker inserts the corresponding number of `nop` instructions in his/her own attack codes. Fig. 3 shows the attack results obtained by a FLUSH+RELOAD attacker in these three situations. It can be seen that an attacker can also capture some doubling and addition operations. But the precision is further reduced due to the series of inserted `nop` instructions. We can infer from Fig. 3 that the partial digits of k are '01101100.....' (S=64), '11001111.....' (S=128), '11111110.....' (S=192). Obviously, the digits of k inferred from the results are not entirely correct. As we set k as '010101.....', there is at least one '0' lost between consecutive '1' bits, and there is at least one '1' lost between consecutive '0' bits. The more consecutive '0' or '1' means the more bits are lost. According to our preliminary analysis of the attack results, when S=64, 128, and 192, the attacker lost at least around 26%, 40%, and 47% of the digits.

To enhance security, we can raise the value of S, however, it could lead to more performance overhead and false positives. Additionally, we can introduce extra checks, such as looking for a series of `nop` instructions after identifying T_1 to detect malicious behaviors. All in all, we set S as 128 for TF-Timer to balance security and efficiency. Note that the flexibility of TF-Timer allows us to extend TF-Timer timely to defend against potential evasion tactics in the future.

C. Fuzzing guest_TSC

As discussed in Section III-C2, the practical implementation of the fuzzer should give special considerations to the following two technical details of utmost importance.

Obfuscate silently. The fuzzer obfuscates the TSC silently. Specifically, we use `get_random_bytes()` to obtain an N-bit random number and subtract it from the *guest_TSC*. Then, TF-Timer stores the new *guest_TSC* value to EDX and EAX registers for the guest VM. In this way, TF-Timer can silently provide an obfuscated TSC value to the VM and it is difficult for an attacker to be aware of it. In addition, TF-Timer's scope of fuzzing is narrow (only T_1), making it more efficient and focused. Therefore, TF-Timer is more concealed and hardly impacts other benign processes.

Maintain properties of `rdtsc/rdtscp`. As Section III-C mentions, there are four properties that any adjustment to the `rdtsc/rdtscp` must take into account.

① **Incremental**. TF-Timer only obfuscates the return value of T_1 by subtracting an N-bit random number. Therefore, the return value of T_1 will be smaller than that of the next timing instruction. Consequently, the return value of the `rdtsc/rdtscp` instruction is strictly increasing. ② **Unpredictable.** TF-Timer obfuscates the return value of T_1 by subtracting a random number. Thus the least significant bits of the return value are still unpredictable. ③ **Differential.** Taking advantage of the inspector, TF-Timer does not change the return value of `rdtsc/rdtscp` instruction in benign processes, thus the cycle difference between two benign calls of the `rdtsc/rdtscp` instruction can be reflected accurately. ④ **Accurate.** TF-Timer only obfuscates the T_1 in a potential attack, thus the timing instructions used by a benign process can accurately reflect the number of cycles the processor has been running.

Considering the above properties, to ensure the security of sensitive data as much as possible and adjust $TV_2 - TV_1$ to a moderate value, similar with [3], we finally set N to 12 in our prototype implementation of TF-Timer.

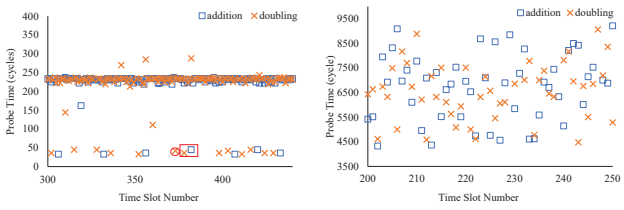
V. Evaluation

We evaluate our prototype of TF-Timer on a Dell PowerEdge R720 with two Intel Xeon E5-2609 (1.90GHz) processors. The guest VM has two vCPUs and runs Ubuntu 18.04.5 LTS. The Linux kernel v4.15, which we patched to install TF-Timer, is used by the host OS. We install Qemu-kvm v2.11.1 for the host OS to manage VMs. Less than 500 lines of new code are added to the Linux kernel.

A. Security Evaluation

Configurations of evaluations. To verify the effectiveness of TF-Timer, we first evaluate the typical timing-dependent attack, FLUSH+RELOAD, by utilizing the widely used toolkit: Mastik [20]. FLUSH+RELOAD targets the ECDSA signature algorithm in OpenSSL 1.1.0h. The experimental environment is similar to that described in Section IV-B. The victim repeatedly calculates ECDSA signatures and the ephemeral key (k) is also set as '010101.....'. The attacker detects the activities of the target cache lines related to point addition and point doubling.

FLUSH+RELOAD attack resistance. First, in native Linux, we implemented the FLUSH+RELOAD attack and obtained the results observed by the attacker. The access pattern of the victim to the target cache lines can be seen in Fig. 4a, which presents the sequence of point addition and point doubling executed by the victim clearly. A shorter probe time in the figure indicates a cache hit, which implies that the victim has just accessed the cache line, i.e., the victim has performed the operation corresponding to this cache line. Conversely, a longer probe time indicates that the victim has not just accessed the cache line. As shown in Fig. 4a, the point doubling in the red circle indicates a '0' of k; the point addition and the point doubling in the red box indicate a '1' of k. In the Linux with TF-Timer, the cache activities obtained by a FLUSH+RELOAD attacker are shown in Fig. 4b. It can be clearly seen that it is difficult for the attacker to distinguish the activities of the target cache lines, thus making it hard to infer the sensitive information. Therefore, TF-Timer shows great mitigation effect on FLUSH+RELOAD attacks.



(a) Native Linux. (b) Linux with TF-Timer.

Fig. 4: The results of FLUSH+RELOAD attack in different scenes.

Other attacks resistance. We also evaluate the effect of TF-Timer against other types of timing-dependent attacks: FLUSH+FLUSH and PREFETCH+PREFETCH attacks. We use Mastik to launch a FLUSH+FLUSH attack in native Linux and in Linux with TF-Timer. We also use the source code from [17] to launch a PREFETCH+PREFETCH attack in native Linux and in Linux with TF-Timer. Our experiments show that in the native Linux, the FLUSH+FLUSH and PREFETCH+PREFETCH attacks operate successfully. However, in the Linux with TF-Timer, TF-Timer can identify the FLUSH+FLUSH and PREFETCH+PREFETCH attackers in time and blur the return value of `rdtsc/rdtscp`. Hence, the attacker is unable to obtain the victim's sensitive information.

TF-Timer is effective against timing-dependent attacks, yet it may not be suitable for countering timing-independent attacks, such as PRIME+ABORT. Nevertheless, it's worth noting that as far as we know, PRIME+ABORT doesn't seem to be successfully executed on modern processors [19]. For now, the predominant cache side-channel attacks primarily fall under the category of timing-dependent attacks, further highlighting the feasibility and wide applicability of TF-Timer.

False positives. To evaluate the false positives of TF-Timer, we deployed the TF-Timer on a host along with a guest VM. We imitate a normal user to use the guest VM for handling daily office tasks (e.g., email communication and website browsing) for a month and log the system activities that use `rdtsc/rdtscp` instructions. Through careful observation of the logs, we did not detect any false positives.

B. Performance Evaluation

To evaluate the overhead induced by TF-Timer to the system, we start a VM and run it for a period of time. After the system activities are relatively stable, we run the SPEC CPU 2006 benchmark three times and calculate the average performance loss induced by TF-Timer. We measure SPEC CPU 2006 scores on the host OS with TF-Timer and without TF-Timer, respectively. From the experimental results shown in Fig. 5, we can see that the overhead induced by the TF-Timer is very tiny. Specifically, the average performance loss is 0.0032%. We also perform a multi-threaded performance evaluation of TF-Timer using PARSEC 3.0 benchmark [21] with the same experimental environment. As shown in Fig. 6, the performance overhead induced by TF-Timer is about 1.01% on average.

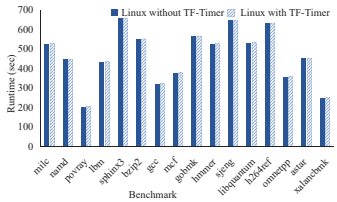


Fig. 5: Performance overhead in SPEC CPU 2006.

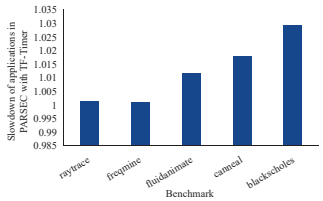


Fig. 6: Performance overhead in PARSEC 3.0.

VI. Conclusion

Cache side-channel attacks pose a significant security threat, particularly in the cases of cloud applications. This paper delves into the analysis and modeling of cache side-channel attacks that leverage `rdtsc/rdtscp` instructions for timing microarchitectural events. We introduce the TMT behavior pattern and propose TF-Timer, a solution that achieves transparency, accuracy, and efficiency by identifying TMT patterns and fuzzing TSC values at the VMM layer. We provide the details of the implementation and evaluation of TF-Timer, demonstrating its effectiveness in mitigating TMT attacks with minimal overhead.

References

[1] G. I. Apecechea, T. Eisenbarth, and B. Sunar, "S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing - and Its Application to AES," in *IEEE S&P*, 2015, pp. 591–604.

[2] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security*, 2014, pp. 719–732.

[3] W. Liu, D. Gao, and M. K. Reiter, "On-Demand Time Blurring to Support Side-Channel Defense," in *ESORICS*, 2017, pp. 210–228.

[4] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache attack," in *DIMVA*, 2016, pp. 279–299.

[5] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-Based Defenses Against Cross-VM Side-Channels," in *USENIX Security*, 2014, pp. 687–702.

[6] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed Inclusion Caches for Mitigating LLC Side-Channel Attacks," in *DAC*, 2017, pp. 1–6.

[7] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Lucky 13 Strikes Back," in *ASIACCS*, 2015, pp. 85–96.

[8] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-Tenant Side-Channel Attacks in PaaS Clouds," in *CCS*, 2014, pp. 990–1003.

[9] Y. Feng, Z. Zhu, S. Li, B. Liu, H. Wang, and D. Meng, "Constant-Time Loading: Modifying CPU Pipeline to Defeat Cache Side-Channel Attacks," in *TrustCom*, 2021, pp. 1132–1140.

[10] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-Channel Attacks," in *ISCA*, 2012, pp. 118–129.

[11] Z. Mi, H. Chen, Y. Zhang, S. Peng, X. Wang, and M. K. Reiter, "CPU Elasticity to Mitigate Cross-VM Runtime Monitoring," *IEEE TDSC*, vol. 17, no. 5, pp. 1094–1108, 2018.

[12] B. C. Vattikonda, S. Das, and H. Shacham, "Eliminating Fine Grained Timers in Xen," in *CCS Workshop*, 2011, pp. 41–46.

[13] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *CT-RSA*, D. Pointcheval, Ed., 2006, pp. 1–20.

[14] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-Level Cache Side-Channel Attacks are Practical," in *IEEE S&P*, 2015, pp. 605–622.

[15] A. Purnal, F. Turan, and I. Verbauwhede, "Prime+Scope: Overcoming the Observer Effect for High-Precision Cache Contention Attacks," in *CCS*, 2021, pp. 2906–2920.

[16] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "RELOAD+REFRESH: Abusing Cache Replacement Policies to Perform Stealthy Cache Attacks," in *USENIX Security*, 2020, pp. 1967–1984.

[17] Y. Guo, A. Zigerelli, Y. Zhang, and J. Yang, "Adversarial Prefetch: New Cross-Core Cache Side Channel Attacks," in *IEEE S&P*, 2022, pp. 1458–1473.

[18] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX," in *USENIX Security*, 2017, pp. 51–67.

[19] R. Zhang, T. Kim, D. Weber, and M. Schwarz, "(M)WAIT for It: Bridging the Gap between Microarchitectural and Architectural Side Channels," in *USENIX Security*, 2023.

[20] Y. Yarom, "Mastik: A micro-architectural side-channel toolkit," 2016.

[21] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *PACT*, 2008, pp. 72–81.