electronics

MDPI

Article

RT-Sniper: A Low-Overhead Defense Mechanism Pinpointing Cache Side-Channel Attacks

Minkyu Song [1,†], Junyeon Lee [2,†], Taeweon Suh [1] and Gunjae Koo [1,*]

[1] Department of Computer Science and Engineering, Korea University, 145, Anam-ro, Seongbuk-gu, Seoul 02841, Korea; minkyu0_song@korea.ac.kr (M.S.); suhtw@korea.ac.kr (T.S.)
[2] Samsung Advanced Institute of Technology, Samsung Electronics, 130, Samsung-ro, Yeongtong-gu, Suwon-si 16678, Korea; junyeon2.lee@samsung.com
* Correspondence: gunjaekoo@korea.ac.kr
† These authors contributed equally to this work.

Abstract: Since cache side-channel attacks have been serious security threats to multi-tenant systems, there have been several studies to protect systems against the attacks. However, the prior studies have limitations in determining only the existence of the attack and/or occupying too many computing resources in runtime. We propose a low-overhead pinpointing solution, called RT-Sniper, to overcome such limitations. RT-Sniper employs a two-level filtering mechanism to minimize performance overhead. It first monitors hardware events per core and isolates a suspected core to run a malicious process. Then among the processes running on the selected core, RT-Sniper pinpoints a malicious process through a per-process monitoring approach. With the core-level filtering, RT-Sniper has an advantage in overhead compared to the previous works. We evaluate RT-Sniper against Flush+Reload and Prime+Probe attacks running SPEC2017, LMBench, and PARSEC benchmarks on multi-core systems. Our evaluation demonstrates that the performance overhead by RT-Sniper is negligible (0.3% for single-threaded applications and 2.05% for multi-threaded applications). Compared to the previous defense solutions against cache side-channel attacks, RT-Sniper exhibits better detection performance with lower performance overhead.

Keywords: malware detection; cache side-channel attacks; security; overhead

Citation: Song, M.; Lee, J.; Suh, T.; Koo, G. RT-Sniper: A Low-Overhead Defense Mechanism Pinpointing Cache Side-Channel Attacks. *Electronics* **2021**, *10*, 2748. https://doi.org/10.3390/electronics10222748

Academic Editor: Krzysztof Szczypiorski

Received: 12 October 2021
Accepted: 6 November 2021
Published: 10 November 2021

1. Introduction

Defenses against malicious processes are critical tasks since cache side-channel attacks have been serious security threats to multi-tenant server systems like cloud infrastructures. The attackers that exploit such attack mechanisms can leak secret data by decoding the changes in shared caches. Generally, the attackers exploit the timing differences in accessing cached and non-cached blocks to decode secret information from the caches. Note that those cached/non-cached blocks are caused by the data accesses of victim processes, thus the attackers can figure out how the victim processes change the cache states, especially by the victim's secret data. Researchers have revealed several attack mechanisms that exploit popular shared libraries and/or applications. For instance, cache side-channel attacks can be performed on cryptography libraries [1–9], well-known user interfaces [10], graphics libraries [11], and autonomous vehicle applications [12]. Recently several researchers disclosed that attackers can access privileged kernel spaces or secret data using the cache side-channel attacks that exploit speculative executions of modern out-of-order processors [13,14]. In order to change the cache state, such attacks first access kernel spaces or secret data using the instructions executed speculatively (i.e., victim codes). Note that those victim codes are executed speculatively since the predictions in the preceding instructions are not resolved yet (Spectre-like attacks), or the preceding instructions may cause faults (Meltdown-like attacks). Then the attackers decode the cache state changes using the cache timing side-channels. Since the first disclosure of Spectre and Meltdown in 2018,

researchers have revealed several variants of the transient execution attacks [15–18]. Most of the modern processors that rely on out-of-order processing and speculative executions are vulnerable to such attacks [19].

Hardware-based defenses can be fundamental defense approaches against cache side-channel attacks, since such attacks exploit the microarchitectural vulnerabilities and cache side-channels. Researchers proposed cache partitioning approaches that strictly prohibit cache sharing between privileged and normal processes [20–24]. Several researchers presented cache randomization methods that dynamically randomize cache tags or replacement mechanisms among different processes [25–27]. Such approaches require hardware modifications in a cache hierarchy. CPU vendors such as Intel developed several hardware patches to defend against the transient execution attacks, and these patches are applied to Intel's new CPU architectures [28]. Even though hardware-based approaches are fundamental defenses against the attacks, such ideas can be applied only to new CPU architecture, thus the computer systems that equip old CPUs are still under severe security threats from the cache side-channel attacks. For instance, many server systems still deploy old CPU architectures such as Intel Haswell and Skylake due to cost issues, thus software patches or software-based defenses are the only available solutions for such systems.

Software-based approaches can make use of hardware performance counters (HPCs) to detect cache side-channel attacks in runtime. HPCs are hardware registers that count CPU or memory-related events in a processor. Since the common cache side-channel attacks execute memory-related instructions frequently to manipulate the target cache states, the attacks may provoke uncommon hardware performance counts such as skyrocketing cache misses. The signature-based detection methods [6,29–34] rely on the models with HPCs collected from several attacks such as Flush+Reload [1], Prime+Probe [5], and Flush+Flush [6]. The anomaly-based detection methods [35–39] developed the detection model based on the normal behavior of the potential victim applications, such as cryptography applications. The proposed solutions may detect the existence of malicious processes effectively; however, pinpointing malicious processes exhibits significant performance overhead, which is one of the critical hurdles for employing software-based solutions.

In this paper, we propose a lightweight pinpointing solution called Run-Time (RT)-Sniper. Before an attack is executed, RT-Sniper keeps monitoring the HPCs of each physical core. When suspicious activity is detected on a specific core, RT-Sniper changes the monitoring resolution to a process-level. Monitoring HPCs per core causes negligible performance overhead compared to the previous pinpointing methods. To speed up the detection latency on a multi-core system, we deploy multi-sentinels to the cores, which enables faster and lighter monitoring than a single-threaded approach. RT-Sniper uses a modified cumulative sum (CUSUM)-based algorithm for detecting malicious processes. Each sentinel calculates the suspect score using the selected hardware performance metrics and accumulates the score in every collecting round.

The rest of the paper is organized as follows. We introduce the attack mechanisms of cache side-channel attacks and the defense approaches against the attacks in Section 2. The detailed detection mechanism of RT-Sniper is described in Section 3. We exhibit evaluation results of RT-Sniper in Section 4 and conclude this paper in Section 5.

2. Background and Related Work

2.1. Cache Side-Channel Attacks

The attack processes that exploit cache side-channel attacks usually aim at cache resources shared by victim processes. Note that caches are shared components in a processor. A modern high-performance processor usually includes a large last-level cache (LLC) shared by multiple cores, thus this LLC is shared by multiple active processes running on the multiple cores. Each core has private level-1 (L1) and level-2 (L2) caches; however, these private caches can be also shared by multiple processes if the processor core supports multithreading. In order to read out secret information, the attack processes typically deploy two different steps: initialization and verification phases. During the initialization

phase, the attack processes manipulate the status of cache blocks which may be touched by victim processes. Note that the attackers know the manipulated cache states, thus they can later detect the changes by sensitive data or instructions of the victim processes. Most of the attacks remove the target cache blocks from the cache. One exception is a RELOAD+REFRESH attack, which keeps the target cache blocks in the cache in order to exploit the aging information of the target cache blocks [7]. The attackers can utilize two different approaches for evicting the existing data from the target cache blocks. The first approach is using the explicit cache manipulating instructions such as *clflush* of the x86 instruction set architecture (ISA) [1,6]. The attacker can also utilize the indirect methods that create conflicts in the target cache blocks by using an eviction set [5,8,9]. An eviction set means a set of data that can occupy all or a part of cache blocks in target cache sets. The attacker manipulates the addresses of an eviction set data to occupy target cache sets. When the attacker loads the eviction set, the cache blocks in the target cache sets are allocated by the eviction set data, thus the old data that has resided in the target sets are evicted from the cache.

During the verification phase, the attackers decode secret information by checking whether the target cache blocks were touched by the victims. The most common approach is measuring data access latency, since attackers can observe different cache access time for cache hits and misses. For instance, if an attack process exploits a last-level cache (LLC), the attacker figures out whether the target cache block is in LLC (i.e., cache hit) or in main memory (i.e., cache miss) by measuring the access time to the target data. Researchers revealed various attack approaches that exploit the cache timing side-channels. FLUSH+RELOAD and similar attacks [1,8] monitor the data loading latency for the target cache block. PRIME+PROBE [5] also exploits differences in access latency to caches but in an eviction set. RELOAD+REFRESH [7] also relies on differences in cache access time considering the known cache replacement policies. EVICT+TIME [9] detects the timing differences while performing cache eviction processes. The attackers implementing such attack approaches are able to measure and compare the cache access latency using the instructions (*rdtsc* of x86 ISA) that read a timestamp counter.

2.2. Detection of Cache Side-Channel Attacks

As the cache side-channel attacks have been severe threats to secure systems, researchers proposed several run-time detection approaches that utilize hardware performance counters (HPCs) [6,29–40]. HPCs are hardware registers that count architectural events such as execution cycles, cache hits/misses, branch prediction hits/misses, and so on. Since the cache side-channel attacks provoke repeated cache hits/misses within a short time period to examine the cache state changes, such attacks can be identified by monitoring dramatic changes in HPCs of the related hardware events. Thus, the run-time detectors collect performance data from HPCs to identify performance value changes created by malicious processes. We can categorize the run-time detection methods that utilize HPCs as three types: anomaly-based detection, signature-based detection, and hybrid detection mechanisms that combine the former two approaches.

Anomaly-based mechanisms: The anomaly-based detection mechanisms utilize the performance data studied from potential victim applications such as comparison baselines to detect the *abnormal* behaviors caused by attacker processes. Thus, such approaches can detect the anomalies of performance values by unknown attacks. However, those defenses can protect only previously studied victim applications. Spydetector [35] detects cache side-channel attacks by clustering the real-time activity of the victim process. Under the attack, the victim's performance is affected by contentions in caches, and it leads to a spike in workload. It can detect Flush+Flush, Flush+Reload, and Prime+Probe attacks with higher than 95% of accuracy under no stress conditions. Briongos et al. proposed CacheShield so that can protect several cryptography functions such as RSA and AES [36]. CacheShield studies the performance counter data of the cryptography applications to be protected. CacheShield monitors the behavior of the protected applications in run-time,

then it figures out whether attacks are fulfilled by detecting the abnormal performance data changes. Mushtaq et al. proposed a machine-learning-based runtime detection mechanism called Whisper [37]. Whisper applies the three machine-learning models (decision tree, SVM, and random forest) for the sampled hardware performance data. Whisper can detect cache side-channel attacks with 98% accuracy and an average of 10.8% of performance overhead when the fine-grained sampling is applied. The detection accuracy decreases to 97% for the coarse-grained sampling mode; however, the performance overhead by Whisper is still 3.4%.

Signature-based mechanisms: The signature-based detection methods exploit the unique performance counter features caused by the known cache side-channel attacks. For instance, Flush+Reload [1] causes a lot of cache misses in a short time because it rapidly repeats flushing and accessing the same cache line. Then a detector modifies the threshold of cache miss per unit time to detect the Flush+Reload. Payer proposed the signature-based detection mechanism called HexPADS [29]. HexPADS collects the selected hardware performance counts at the process-level to compare the collected data with the predefined threshold levels. HexPADS slows down the attack mechanisms of attack processes by making the target core busy. HexPADS deploys a coarse-grained sampling to minimize performance overhead. However, the long detection latency of HexPADS means that part of the secret data can be leaked. Depoix et al. [30] proposed a convolutional neural network (CNN) model to detect malicious processes. The proposed mechanism collects hardware events from the running processes with 100 ms of sampling frequency and then evaluates the collected events using the CNN model to detect malicious processes. The authors reported that the detection accuracy is 97% on average based on the F-score; however, the runtime detection rates are not provided in the paper.

Hybrid mechanisms: Several researchers proposed hybrid approaches that combine the above two different detection mechanisms. CloudRadar employs a threshold-based detection approach that compares the calculated threshold values with the hardware performance counts [31]. In order to compute the threshold levels, CloudRadar uses nine normal applications (i.e., anomaly-based) and two attack examples (i.e., signature-based). CloudRadar achieves 80% detection accuracy under the fine-grained sampling setting. However, when configured with the coarse-grained sampling, it exhibits at least a 20% false-positive rate, which is a significant limitation. CBA-Detector deploys a self-feedback mechanism to reduce a false-positive rate [32]. CBA-Detector first detects the cache side-channel attacks using a machine-learning-based detection mechanism, then it validates the detection results using Pintool. Pintool is a dynamic instruction instrumental tool that can analyze the executed instruction in runtime. With the self-feedback mechanism, CBA-Detector decreases the false-positive rate by less than 50%; however, this mechanism causes significant performance loss.

2.3. Limitations

Some of the previously proposed software-based approaches fail to pinpoint malicious processes since the software solutions can just detect the existence of cache side-channel attacks. For instance, several anomaly-based mechanisms [35–37] can only detect whether specific victim processes are under attack or not. Some signature-based mechanisms such as [30,31] can only disclose whether attack processes are running on a system. Note that the software solutions cannot block the attacks without identifying malicious processes.

Another problem of the existing software approaches is the significant performance overhead caused by collecting hardware performance counts. In order to pinpoint malicious processes, the software detector needs to monitor per-process hardware events, thus the detector requires the collecting of hardware performance counts of each process by checking process ids (pids) in runtime. Hence the software defense solutions that can identify malicious processes exhibit significant performance overhead. For instance, HexPADS [29] results in a 20% performance loss if fine-grained sampling (less than 10 ms) is

applied. CBA-Detector [32] reduces the performance overhead using several optimization techniques; however, it still exhibits at least 12% of performance overhead.

3. Materials and Methods

In this paper, we propose RT-Sniper, a software-based defense mechanism against cache side-channel attacks. RT-Sniper employs a two-level detection approach in order to reduce the performance overhead for pinpointing malicious processes. RT-Sniper first detects the core that runs malicious processes using core-level performance counts. After the filtering, it investigates only the processes which are running on the detected core. In order to minimize the performance overhead for collecting performance counts, RT-Sniper applies an optimized filtering mechanism that can adjust the frequency of collecting the process-level performance counts. RT-Sniper also exploits a filtering method based on a cumulative sum to collect the events directly influenced by attack processes. Hence, RT-Sniper can accurately detect attack processes without large overhead using the two-level detection approach. In the following subsections, we will describe the detailed two-level mechanism of RT-Sniper and the selected hardware events which are used for calculating malicious scores.

3.1. Two-Level Pinpointing Mechanism

As mentioned in the previous section, pinpointing malicious processes may cause significant performance overhead to multi-tenant systems. Such heavy processing burden by the software solutions that exploit HPCs has resulted from frequent system calls and directory accesses. In order to identify malicious processes, the software defense solutions collect HPCs of each process. For instance, the detectors can make use of a system call, called *perf_event_open*, provided by Linux operating systems to collect HPCs from CPUs. We can adjust the levels of the performance monitors such as system-level, core-level, and process-level performance counts. In order to collect the process-level HPCs, the detectors need to provide process ids (pid) that can be obtained by scanning */proc* directory. Such a process is a heavy burden for the defense solutions since it requires directory scans and system calls for all running processes.

In order to study the performance burdens from monitoring HPCs, we measure the performance overhead by monitoring levels (i.e., per-core and per-process) and sampling frequencies. For this experiment, we use the computer system that equips a modern 8-core CPU running at 3.6 GHz. The detailed system configuration is described in Section 4. We collect six performance counts: L2 misses, L2 hits, LLC misses, LLC hits, executed instructions, and execution cycles. These data are required to compute the three performance metrics (L2 miss rates, LLC miss rates, and IPCs) used by RT-Sniper as mentioned in Section 3.2. We execute SPEC2017 and PARSEC benchmarks while monitoring HPCs of each core (per-core) and each process (per-process) to measure the performance burdens by monitoring HPCs as shown in Table 1. We also measure the performance overhead by different sampling frequencies (1 ms and 10 ms). Our experiments exhibit that the performance overhead of monitoring HPCs is up to 12.4% for PARSEC benchmarks and 104.5% for SPEC2017 benchmarks. Even though our experiments reveal that the per-process monitoring causes significant performance overhead, the process-level HPCs are essential data for pinpointing malicious processes.

In order to reduce the significant performance overhead by per-process monitoring, RT-Sniper employs a two-phase operation based on a filtering mechanism. RT-Sniper first monitors HPCs of all cores and then selects cores that are suspected of running malicious processes. After the cores that piggyback malicious processes are picked, RT-Sniper pinpoints the malicious processes within the cores by checking per-process HPC data. Note that monitoring per-core HPCs does not cause severe performance loss as shown in the fourth column of Table 1. Furthermore, RT-Sniper can examine HPCs of the several processes running on the selected cores to reduce the performance burdens from per-process monitoring. Our evaluation reveals the performance overhead by RT-Sniper's

per-core monitoring is negligible, which is 0.37% and 1.67% with 10 ms and 1 ms sampling period respectively.

Table 1. Overhead of SPEC and *PARSEC* [(1)] execution time according to HPC collection strategies. (1) streamcluster and netstreamcluster are excluded since the execution time is inconsistent due to frequent synchronizations [41].

Benchmark	Sampling Period	Per-Process	Per-Core	RT-Sniper (Single)	RT-Sniper (Multi)
PARSEC	1 ms	12.4%	1.67%	2.26%	3.76%
	10 ms	5.99%	0.37%	1.51%	2.05%
SPEC2017	1 ms	104.5%	10.0%	18.3%	6.36%
	10 ms	17.74%	5.23%	7.12%	0.74%

For the core-level monitoring operation, RT-Sniper launches multiple processes, called sentinels. RT-Sniper assigns a single sentinel per physical core, thus the number of launched sentinels is equal to the number of physical cores in the monitored system. Each sentinel runs as an independent thread that monitors HPCs of each physical core. Compared to the single-threaded approach that monitors HPCs of all cores in a round-robin fashion, RT-Sniper's multi-threaded approach has advantages for two performance factors: detection latency and overhead. In terms of detection latency, with a 1 ms sampling interval (i.e., 1 ms of sleep period after monitoring HPCs), actual scanning intervals by the single-sentinel and the multi-sentinel approaches measure 1.348 ms and 1.070 ms, respectively. Assuming both approaches require the same number of monitoring rounds until a suspicious core is detected, our measurement reveals the multi-sentinel approach can detect the core that is running an attack process 20% faster than the single-sentinel one. Our measurement reveals that the performance overhead by the multi-sentinel approach is slightly higher and even lower than the single-threaded per-core monitoring approach as shown in the fifth and the sixth column of Table 1. Compared to the single-sentinel approach, the multi-threaded monitoring causes 0.5–1% higher performance overhead for PARSEC benchmarks and 7–12% lower overhead for SPEC2017 benchmarks.

Figure 1 depicts RT-Sniper's two-phase operation for detecting malicious processes. RT-Sniper launches sentinels on each physical core (①) to collect core-level HPCs. Each sentinel periodically collects hardware events to compute core's suspect score based on the collected HPCs. The detailed score computation method will be described in the next section. Then when the score of a certain core is greater than the predefined threshold level, the core is piked for further scrutiny. The sentinel of the selected core then collects HPCs of the processes that are running on the core (②). RT-Sniper calculates the suspect score of each process to block the processes whose score is greater than the process-level threshold.

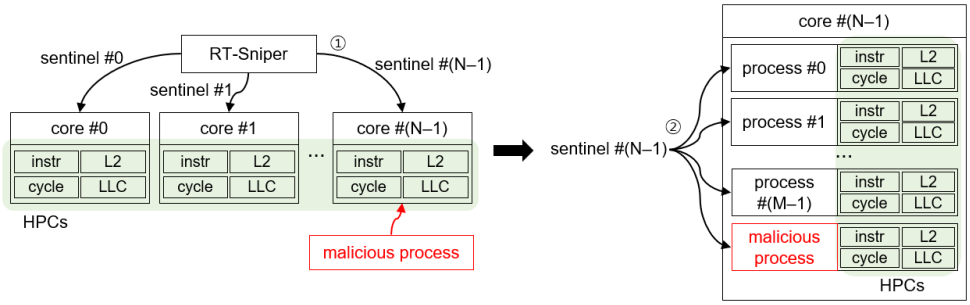

Figure 1. RT-Sniper mechanism: core-level filtering to process-level pinpointing.

3.2. Detection Algorithm

3.2.1. Selected Hardware Performance Counts

In order to calculate the suspect scores, RT-Sniper monitors the several selected HPCs that can effectively reflect the hardware events derived from the executions of cache side-channel attacks. The selected hardware performance metrics are L2 miss rate, last-level

cache (LLC) miss rate, and instructions per cycle (IPC). We observe that these performance metrics are significantly influenced by the popular cache side-channel attacks. In order to compute the performance metrics, RT-Sniper collects six performance counts, such as L2 misses/hits, LLC misses/hits, instructions, and execution cycles. Now we describe how these metrics can effectively reflect the influences from cache side-channel attacks.

L2 miss rates: RT-Sniper uses L2 miss rates of physical cores and running processes to detect cache side-channel attacks, since the attacks exploit an eviction set in LLC [5,7,8]. Figure 2 illustrates how L2 misses are incurred by an LLC eviction set. In this example, we assume the L2 cache is a 2-way set-associative cache with LRU replacement policy and LLC is a 4-way set-associative cache. The first requests for the eviction set (i.e., four blocks from mb1 to mb4) result in cold misses in both L2 and LLC since the requested blocks do not exist in both caches initially. After the first requests for the eviction set, the target cache set in LLC includes all four blocks (mb1–mb4); however, the target set in L2 has only the two latest blocks (mb3 and mb4). In the same way, the subsequent requests for the same eviction set encounter cache hits in LLC and the cache missed in L2. This simple scenario explains why high L2 miss counts are observed when cache side-channel attacks are performed. Considering that LLC has 1.5 to 2 times higher associativity compared to L2 [7], the cache side-channel attacks that exploit cache eviction sets typically exhibit high L2 miss counts.

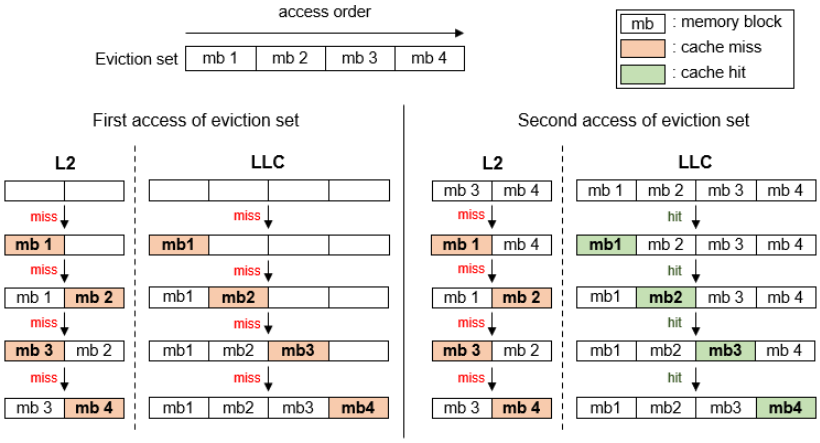

Figure 2. Impact of accessing an eviction set on L2 cache and LLC with LRU replacement policy.

LLC miss rates: RT-Sniper can use high LLC miss rates to detect the cache side-channel attacks that repeatedly access target memory blocks such as Flush+Reload [1]. Such types of attacks periodically flush cache blocks from all levels of caches and load the memory blocks, thus high cache miss rates are observed from all levels of caches. Therefore, RT-Sniper uses unusual high LLC miss rates as one of the indicators of such attacks.

IPC: Cache side-channel attacks usually provoke more frequent L2 and/or LLC misses while the attacks are performed, thus the IPC of the attack processes can be lower than normal applications. Hence, RT-Sniper uses the IPC as another barometer to detect the existence of the attacks.

3.2.2. Detection Algorithm

In order to detect cache side-channel attacks, RT-Sniper applies a cumulative sum algorithm (CUSUM) using the selected performance metrics [42,43]. RT-Sniper employs CUSUM since it is effective in analyzing time-series data using a simple recursive approach. At every sampling point, RT-Sniper computes the suspect score of the current state and then accumulates the current score to the previous cumulative score. If the cumulative score exceeds the predefined threshold level, RT-Sniper determines that a malicious process is running. The equation of CUSUM is shown as follows.

$$\begin{cases} S_0 = 0 \\ S_n = max(0, S_{n-1} + s_n - k) \\ if S_n > th, \rightarrow \text{detection result is alarmed} \end{cases} \quad (1)$$

In the above equation S_n is a cumulative score of state n and s_n is a partial score of state n. k represents a shift constant for normal correction. th is a threshold value that is used for comparing with the computed cumulative score to detect malicious processes. The partial score s_n of the current state is computed using the hardware performance metrics highly influenced by cache side-channel attacks. In the previous section we selected the three performance metrics: L2 miss rates, LLC miss rates, and IPC. Using these metrics, s_n is computed as follows. In the below equation $l2\_mr_n$ and llc_mr_n represent a L2 miss rate and a LLC miss rate of state n respectively. Note that malicious processes usually exhibit high s_n value.

$$s_n = weight\_L2 \times l2\_mr_n + weight\_LLC \times llc\_mr_n + weight\_IPC \times (1/ipc_n) \quad (2)$$

Algorithm 1 shows the pseudo-code of the modified CUSUM algorithm. RT-Sniper initializes the required parameters (weight_L2, weight_LLC, weight_IPC, and age) before launching the monitoring processes (i.e., sentinels) as shown in line 1. RT-Sniper loads the previously computed score (line 3). Then RT-Sniper computes the current suspect score (lines 4–7) and adds the current score to the loaded accumulated score (line 8). RT-Sniper saves the accumulated score for the next sampling period. RT-Sniper's parameters (i.e., weights, threshold, and age of the detection algorithm) are determined by training. We define the cost function that uses RT-Sniper's parameters as input arguments. Note that the cost is a combination of RT-Sniper's performance metrics: detection accuracy, detection latency, and false-positive rate. We use a popular least-square algorithm to find the optimal parameters that can make the minimum cost.

Algorithm 1 Pseudo-code of Computing CUSUM by RT-Sniper

```
1:  predefined parameters : weight_L2, weight_LLC, weight_ipc, age
2:  procedure CALCULATE_CUMULATIVE_WEIGHTED_SUM(id, HPC)
3:      accum_score ← get_accumulated_score_from_id(id)
4:      weighted_L2 = weight_L2 × l2_miss_ratio
5:      weighted_LLC = weight_LLC × llc_miss_ratio
6:      weighted_IPC = weight_IPC × (1 / ipc)
7:      cur_score = weighted_L2 + weighted_LLC + weighted_IPC − age
8:      accum_score = accum_score + cur_score
9:      save_accumulated_score_to_id(id, accum_score)
10:     return accum_score
11: end procedure
```

4. Evaluation Results

We evaluated RT-Sniper on Ubuntu 18.04.4 LTS (kernel 5.3.0-62), which is running on a system that equips a 3.6 GHz 8-core Intel i9-9900K CPU with 32 GB main memory and a 512 GB solid-state drive (SSD). The cache configuration of i9-9900K is summarized in Table 2. We tested RT-Sniper with two sampling periods: fine-grained sampling (1 ms) and coarse-grained sampling (10 ms).

Table 2. Cache Configuration of 8-core Intel i9-9900K.

Cache	Size	Set Associativity
L1 instruction	32 KiB	8-way
L1 data	32 KiB	8-way
L2	256 KiB	4-way
L3 (LLC)	16 MiB	16-way

In order to evaluate the detection performance of RT-Sniper, we use four performance criteria: detection accuracy, detection latency, false-positive rate, and performance overhead.

1. Detection accuracy represents how a detector successfully detects running malicious processes. The detector must be able to distinguish malicious processes from many normal processes. We calculated the accuracy as (the number of detected attacks/the number of attack attempts).
2. Detection latency means how quickly the attack is detected. There are two methods to calculate. The first one is an elapsed time from the attack launch to detection. The second one is a ratio of elapsed time for detection to the attack execution time.
3. False-positive rate indicates how many normal processes were falsely identified as attack processes. We represent the false-positive rate as (the number of false-positive/the execution time).
4. Performance overhead means how much a detector influences overall system performance. We calculate the performance overhead as (the execution time with detector/the execution time without detector).

4.1. Accuracy and Latency

We measure the detection accuracy and the detection latency of RT-Sniper using the popular cache side-channel attacks and victim applications. As previous research disclosed how the cache side-channel attacks can leak secret data (e.g., encryption keys) from the encryption applications, we also study the similar attacker-victim models to evaluate RT-Sniper. We select four attacker applications that implement Flush+Reload type attacks [44,45] and Prime+Probe type attacks [46,47]. Note that Flush+Reload and Prime+Probe are attack mechanisms frequently employed by popular cache side-channel attacks, including the recent Spectre and Meltdown PoC codes. We use the AES encryption application that is included in OpenSSL 3.0.0 as a victim application. AES encrypts 16-byte plain texts generated randomly using a 32-byte encryption key. We measured the detection latency of RT-Sniper by measuring the elapsed time between when the attack process is launched and when RT-Sniper pinpoints the attack process. We observed that the attack took approximately 900 ms on our testbed for extracting the 256-bit encryption key. We assume the detection by RT-Sniper is successful if the attack process is pinpointed within 100 ms from launching, which is equivalent to the time required for extracting around 10% of total encryption key data. If RT-Sniper takes longer than 100 ms to pinpoint the malicious process, we regard it as a failure. In order to test the detection sensitivity by workload level of a system, we also evaluate the performance of RT-Sniper under various stress levels. Among SPEC2017 benchmarks, we chose eight applications that exhibit various L2/LLC access rates. Based on the number of L2/LLC references per unit time, we classify lbm_r, cam4_r, bwaves_r, and parest_r as average stress (AS) applications and imagick_s, pop2_s, bwaves_s, and nab_s as high stress (HS) applications. We launch the attack process on the same core that is running one of AS or HS applications to make stressed environments. No stress (NS) means only the attack process and the victim application run on the system.

Table 3 shows the experimental results regarding the detection accuracy and latency of RT-Sniper according to sampling periods. With 1 ms fine-grained sampling, the RT-Sniper successfully detected the malicious process with an accuracy of 99.7% on average. Under the various stress conditions, the detection accuracy for Prime+Probe 2 dropped by 1.2–2.2%, while the detection accuracy for the other attacks remains close to perfection.

When it comes to the detection latency, all attacks were detected in 26.4 ms on average. Even though the detection latency is increased as the stress level intensifies, the worst-case latency is within 40 ms. With 10 ms course-grained sampling, RT-Sniper successfully identified the malicious process with a 99.5% accuracy on average. Specifically, it did not fail to detect all of the Flush+Reload attacks with an average latency of 51.3 ms. In the case of Prime+Probe attacks, there are a few failures under the stress conditions, but the accuracy is still more than 98% in the worst case. Detection latency was increased 1.94 times on average under the high stressed condition. Under the high-stressed condition, RT-Sniper fails to detect Prime+Probe type attacks occasionally, although the failure rate is extremely low. Note that RT-Sniper exploits high L2 miss rates to detect Prime+Probe type attacks as mentioned in Section 3.2.1. HS applications can cause high L2 hit rates since such applications access L2 frequently. Therefore, RT-Sniper may fail to select a malicious core using the core-level L2 miss rates due to severe interference from HS applications. Note that RT-Sniper cannot move on the second phase if the process-level filtering fails.

Table 3. Detection accuracy and latency according to the sampling granularity.

Attack Types	Stress Levels	Fine-Grained (1 ms)			Course-Grained (10 ms)		
		Accuracy (%)	Latency		Accuracy (%)	Latency	
			Time (ms)	Ratio (%)		Time (ms)	Ratio (%)
`Flush+Reload 1`	NS	100	20.5	1.01	100	37.9	3.30
	AS	100	22.9	0.76	100	51.2	2.30
	HS	100	36.2	1.19	100	51.5	2.27
`Flush+Reload 2`	NS	100	18.1	1.04	100	32.1	3.30
	AS	100	25.5	0.96	100	34.4	1.78
	HS	100	25.9	0.98	100	36.0	1.86
`Prime+Probe 1`	NS	100	19.3	1.14	100	36.4	3.84
	AS	100	24.8	0.95	99.6	67.5	3.59
	HS	100	25.6	0.96	99.4	67.6	3.49
`Prime+Probe 2`	NS	100	24.9	1.50	100	37.7	3.28
	AS	98.8	35.1	2.18	97.8	76.2	3.44
	HS	97.8	38.5	1.68	98.4	75.0	3.31

4.2. False Positive Rate

In order to evaluate the false-positive rate by RT-Sniper, we executed 43 applications from SPEC2017 and 45 applications from PARSEC repeatedly. The results are shown in Table 4. RT-Sniper incorrectly detected SPEC2017 3.3 times per hour, and PARSEC 2.9 times per hour with fine-grained sampling and SPEC2017 7.3 times per hour, and PARSEC 1.6 times per hour with coarse-grained sampling. In detail, in the case of SPEC2017, 638.imagick_s and 619.lbm_s are the most frequently false-detected applications. As a result of collecting/analyzing the cache events while running them, it was confirmed that they frequently caused the high cache misses for a certain period. The others also caused high cache misses sometimes, but it did not last long enough to be detected as an attack. Note that in Section 4.1, RT-Sniper needs at least 20–50 ms to detect the process as an attack, thus a normal application can be falsely detected if it keeps high cache misses.

Table 4. False-positive rate caused by benchmarks.

	Fine-Grained (1 ms)	Course-Grained (10 ms)
SPEC2017	3.3 fp/h	7.3 fp/h
PARSEC	2.9 fp/h	1.6 fp/h

4.3. Overhead

The overhead was measured by comparing the execution times of benchmarks with and without RT-Sniper. Three benchmarks are used: SPEC2017, LMbench and PARSEC. SPEC2017 and LMbench are single-threaded applications, and PARSEC is composed of multi-threaded applications. *streamcluster* and *netstreamcluster* from PARSEC were excluded in the evaluation because their behaviors were inconsistent due to frequent synchronizations. SPEC2017 was executed three times with the default data sets, and LMbench was executed five times configured with the data size of 8192 MB. PARSEC was executed 10 times using the native input sets with spawning 16 threads.

Figure 3 shows the normalized execution times of three benchmark suites while RT-Sniper is running. With a 1 ms sampling period, LMbench takes 3.82% more time on average. The file-system applications were mostly affected by a 5.3% increase, and the memory-load applications were affected by a 4.7% increase. The SPEC2017 takes 4.8–5.4% more on average. The 10 ms sampling incurs a negligible impact on the execution times of single-threaded benchmarks. The execution time of SPEC2017 and LMbench was increased a less than 1%, even in the worst case. The execution time of multi-threaded PARSEC increased 2.05% on average. We strongly believe that the barriers in PARSEC make the other threads wait until the sentinels are switched out.

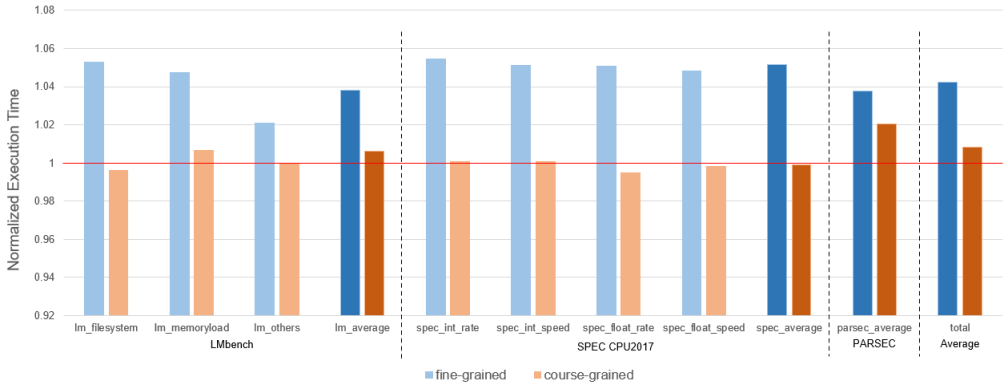

Figure 3. Normalized execution time of SPEC2017, PARSEC and LMBENCH with RT-Sniper.

4.4. Comparative Analysis

Table 5 provides a summary of comparative analysis with the related studies. We summarized this comparison for the evaluation criteria used in this work: detection accuracy, detection latency, false-positive rate, and performance overhead. Note that the works were evaluated with different conditions and purposes. The conditions that affect the performance are listed in the table as much as possible to avoid the confusion of the experiment environment. Spydetector [35] detected more than 95% of the F+R, P+P and F+F. However, it had not been evaluated under high stress conditions. Depending on the data window length, it caused overhead from 0.68% to 2.22%. CacheShield [36] detected 100% of the F+R and P+P within 13.4 ms under various stress conditions. However, it caused 21.4% false positives in the worst case. In addition, it assumed that the detector already knew the pid of the process that operated the cryptography libraries targeted to detect. Thus, in real usage, there is an additional workload to identify the processes and it occurred more overhead. Whisper [37] detected 97.78% of the F+R, P+P and F+F under the concurrent running of the SPEC benchmark. In terms of latency, it had a large spectrum, from 0.21% to 40% completion of attack, which depends on the detected attack. It caused an overhead of about 3.4% with coarse-grained sampling and 10.8% with the fine-grained sampling interval. HexPADS [29] detected 100% of the F+R and only caused less than 1% overhead. However, it had been evaluated under ideal conditions (no stress) and seriously large sampling intervals (>1 s). Depoix et al. [30] provided 97.16% clustering accuracy based on F-score and occupied CPU resources from 4% to 5%. It didn't provide any real-time detection results under stress conditions. CloudRadar [31] presented the ROC of true

positive rate and false-positive rate. Following the authors' arguments, the false positive rate depends on the window size and proper true positive rate. After configuring the true-positive rate as 100%, the false-positive rate was 0% when the window size was 5 and 20% when the window size was 1. Similarly, detection latency also depends on the window size. The overhead was presented as less than 5%, but it had a similar limitation to CacheShield, which assumed that the detector already knew the pid to be protected. CBA-Detector [32] pinpointed more than 98% of F+R, P+P, F+F before the attack stole the full key. Through the improvement of the false-positive feedback mechanism using Pintool, the false-positive rate decreased to 1.3 times per hour. Nevertheless, Pintool occupied an extremely large portion of CPU resources at run-time, and the detector caused 23.9% overhead in the worst case. Compared to those works, RT-Sniper shows at least the second state-of-the-art performance in all metrics. It detected 99.65% of F+R and P+P under the high stress condition within 38.4 ms, and 2.1% of attack completion, and falsely detects the normal benchmarks, SPEC and PARSEC, 3.75 times per hour on average. In terms of overhead, RT-Sniper shows the best performance among the studies about pinpointing [29,30,32]. It even generates less overhead than some anomaly-based detection methods [36,37].

Table 5. Performance of previous works (1) Condition 2 is in which sampling granularity was evaluated. (Fine: less than several millisecond, Course: more than 10 millisecond).

Articles	Target	Stress	Accuracy	Latency	FP (Benchmark)	Granularity (1)	Overhead
spydetector [35]	FR, PP, FF	NA	>95%	N/A	N/A	Course	0.68–2.22%
CacheShield [36]	FR, PP	Y (web app, video, Randmem)	100%	<13.4 ms	<21.4% (web apps)	Fine	<5%
WHISPER [37]	FR, PP, FF	Y (SPEC)	97.76%	0.21–40%	1.2% (SPEC)	Fine, Course	3.4–10.8%
HexPADS [29]	FR	NA	100%	N/A	N/A	Course (1 s)	<1%
Depoix et al. [30]	FR, PP	NA	97.16% (F-score)	N/A	0.33% (F-score)	Course	4–5%
CloudRadar [31]	FR, PP	NA	100%	<5.1 ms	0–20%	Fine	<5%
CBA-Detector [32]	FR, PP, FF	Y (PARSEC)	>98%	10–80%	1.3 fp/h (parsec)	Course	4.8–23.9%
RT-Sniper	FR, PP	Y (SPEC)	99.65%	38.4 ms, 2.1%	3.75 fp/h (spec, parsec)	Fine, Course	2.05–3.76%

5. Conclusions

In this paper, we propose RT-Sniper, a lightweight defense solution that can pinpoint malicious processes effectively. RT-Sniper first monitors hardware performance counters per core to detect the core where malicious processes run by checking the cumulative scores. After this phase, RT-Sniper collects the performance counts of each process that is running on the selected core to pinpoint malicious processes. During the per-core monitoring phase, RT-Sniper monitors the performance counts of each core in parallel in order to reduce detection latency. Note that RT-Sniper can minimize performance overhead since it collects the performance counts at the core-level. Our evaluation exhibits that RT-Sniper can pinpoint malicious processes with 99.7% of accuracy within 26.4 ms on average under wide ranges of stress levels when the fine-grained sampling mode (1 ms sampling) is applied. Using the coarse-grained sampling mode (10 ms sampling), RT-Sniper exhibits nearly the same detection accuracy (99.6%) while the detection latency increases (50.2 ms). The performance overhead by RT-Sniper is pretty low compared to the previously proposed software defense solutions. When the fine-grained sampling mode is configured, RT-Sniper slows down the system performance for single-threaded benchmarks (SPEC2017 and LMBench) and multi-threaded benchmarks (PARSEC) by 4.48% and 3.76%, respectively. The performance overhead by RT-Sniper of the coarse-grained sampling mode is negligible (0.3% for single-threaded applications and 2.05% for multi-threaded applications). Compared to the previous defense solutions against cache side-channel

attacks, RT-Sniper exhibits better detection performance (i.e., detection accuracy, latency, and false-positive rate) with lower performance overhead.

Author Contributions: Conceptualization, M.S., J.L., T.S. and G.K.; Data curation, M.S., J.L.; Formal analysis, M.S., J.L., T.S. and G.K.; Investigation, M.S. and J.L.; Methodology, M.S., J.L., T.S. and G.K.; Project administration, M.S.; Resources, M.S. and J.L.; Software, M.S. and J.L.; Supervision, T.S.; Validation, M.S., J.L., T.S. and G.K.; Visualization, M.S.; Writing—original draft, M.S. and J.L.; Writing—review & editing, T.S. and G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Institute of Information and Communications Technology Planning and Evaluation grant funded by the Korea government (MSIT) (No.2019-0-00533, Research on CPU Vulnerability Detection and Validation / No.2019-0-01343, Regional Strategic Industry Convergence Security Core Talent Training Business / IITP-2021-2020-0-01819, ICT Creative Consilience Program).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Yarom, Y.; Falkner, K. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Proceedings of the 23rd (USENIX Security Symposium (USENIX Security 14), San Diego, CA, USA, 20–22 August 2014; pp. 719–732.
2. Gullasch, D.; Bangerter, E.; Krenn, S. Cache games–bringing access-based cache attacks on AES to practice. In Proceedings of the 2011 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 22–25 May 2011; pp. 490–505.
3. Irazoqui, G.; Inci, M.S.; Eisenbarth, T.; Sunar, B. Wait a minute! A fast, Cross-VM attack on AES. In *International Workshop on Recent Advances in Intrusion Detection*; Springer: Cham, Switzerland, 2014; pp. 299–319.
4. Gülmezoğlu, B.; Inci, M.S.; Irazoqui, G.; Eisenbarth, T.; Sunar, B. A faster and more realistic flush+reload attack on AES. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*; Springer: Cham, Switzerland, 2015; pp. 111–126.
5. Liu, F.; Yarom, Y.; Ge, Q.; Heiser, G.; Lee, R.B. Last level cache side-channel attacks are practical. In Proceedings of the 2015 IEEE Symposium on Security and Privacy, San Jose, CA, USA, 18–20 May 2015; pp. 605–622.
6. Gruss, D.; Maurice, C.; Wagner, K.; Mangard, S. Flush+Flush: A fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*; Springer: Cham, Switzerland, 2016; pp. 279–299.
7. Briongos, S.; Malagón, P.; Moya, J.M.; Eisenbarth, T. Reload+refresh: Abusing cache replacement policies to perform stealthy cache attacks. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Online, 12–14 August 2020; pp. 1967–1984.
8. Lipp, M.; Gruss, D.; Spreitzer, R.; Maurice, C.; Mangard, S. Armageddon: Cache attacks on mobile devices. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 549–564.
9. Osvik, D.A.; Shamir, A.; Tromer, E. Cache attacks and countermeasures: the case of AES. In *Cryptographers' Track at the RSA Conference*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1–20.
10. Zhang, K.; Wang, X. Peeping Tom in the Neighborhood: Keystroke Eavesdropping on Multi-User Systems. In Proceedings of the USENIX Security Symposium, Montreal, QC, Canada, 10–14 August 2009; p. 23.
11. Wang, D.; Neupane, A.; Qian, Z.; Abu-Ghazaleh, N.B.; Krishnamurthy, S.V.; Colbert, E.J.; Yu, P. Unveiling your keystrokes: A Cache-based Side-channel Attack on Graphics Libraries. In Proceedings of the NDSS, San Diego, CA, USA, 24–27 February 2019.
12. Luo, M.; Myers, A.C.; Suh, G.E. Stealthy tracking of autonomous vehicles with cache side channels. In Proceedings of the 29th USENIX Security Symposium (USENIX Security 20), Online, 12–14 August 2020; pp. 859–876.
13. Kocher, P.; Horn, J.; Fogh, A.; Genkin, D.; Gruss, D.; Haas, W.; Hamburg, M.; Lipp, M.; Mangard, S.; Prescher, T.; Schwarz, M.; Yarom, Y. Spectre attacks: Exploiting speculative execution. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–22 May 2019; pp. 1–19.
14. Lipp, M.; Schwarz, M.; Gruss, D.; Prescher, T.; Haas, W.; Fogh, A.; Horn, J.; Mangard, S.; Kocher, P.; Genkin, D.; Yarom, Y.; Hamburg, M. Meltdown: Reading kernel memory from user space. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 973–990.
15. Canella, C.; Genkin, D.; Giner, L.; Gruss, D.; Lipp, M.; Minkin, M.; Moghimi, D.; Piessens, F.; Schwarz, M.; Sunar, B.; Bulck, J.V.; Yarom, Y. Fallout: Leaking data on meltdown-resistant cpus. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019.
16. Schaik, S.V.; Milburn, A.; Österlund, S.; Frigo, P.; Maisuradz, G.; Razavi, K.; Bos, H.; Giuffrida, C. RIDL: Rogue in-flight data load. In Proceedings of the 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 20–22 May 2019.
17. Bulck, J.V.; Minkin, M.; Weisse, O.; Genkin, D.; Kasikci, B.; Piessens, F.; Silberstein, M.; Wenisch, T.F.; Yarom, Y.; Strackx, R. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 991–1008.

18. Schwarz, M.; Lipp, M.; Moghimi, D.; Bulck, J.V.; Stecklina, J.; Prescher, T.; Gruss, D. ZombieLoad: Cross-privilege-boundary data sampling. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, London, UK, 11–15 November 2019; pp. 753–768.
19. Canella, C.; Bulck, J.V.; Schwarz, M.; Lipp, M.; Berg, B.V.; Ortner, P.; Piessens, F.; Evtyushkin, D.; Gruss, D. A systematic evaluation of transient execution attacks and defenses. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019.
20. Qureshi, M.K.; Patt, Y.N. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In Proceedings of the 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), Orlando, FL, USA, 9–13 December 2006; pp. 423–432.
21. Kim, T.; Peinado, M.; Mainar-Ruiz, G. STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud. In Proceedings of the 21st USENIX Security Symposium (USENIX Security 12), Bellevue, WA, USA, 8–10 August 2012; pp. 189–204.
22. Shi, J.; Song, X.; Chen, H.; Zang, B. Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring. In Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), Hong Kong, China, 27–30 June 2011; pp. 194–199.
23. Ye, Y.; West, R.; Cheng, Z.; Li, Y. Coloris: a dynamic cache partitioning system using page coloring. In Proceedings of the 2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT), Edmonton, AB, Canada, 23–27 August 2014; pp. 381–392.
24. Liu, F.; Ge, Q.; Yarom, Y.; Mckeen, F.; Rozas, C.; Heiser, G.; Lee, R.B. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In Proceedings of the 2016 IEEE international symposium on high performance computer architecture (HPCA), Barcelona, Spain, 12–16 March 2016; pp. 406–418.
25. Wang, Z.; Lee, R.B. New cache designs for thwarting software cache-based side channel attacks. In Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, CA, USA, 9–13 June 2007; pp. 494–505.
26. Qureshi, M.K. Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping. In Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Fukuoka, Japan, 20–24 October 2018; pp. 775–787.
27. Werner, M.; Unterluggauer, T.; Giner, L.; Schwarz, M.; Gruss, D.; Mangard, S. Scattercache: Thwarting cache attacks via cache set randomization. In Proceedings of the 28th USENIX Security Symposium (USENIX Security 19), Santa Clara, CA, USA, 14–16 August 2019; pp. 675–692.
28. Affected Processors: Transient Execution Attacks & Related Security Issues by CPU. Available online: https://software.intel.com/content/www/us/en/develop/topics/software-security-guidance/processors-affected-consolidated-product-cpu-model.html (accessed on 1 November 2021).
29. Payer, M. HexPADS: A platform to detect "stealth" attacks. In *International Symposium on Engineering Secure Software and Systems*; Springer: Cham, Switzerland, 2016; pp. 138–154.
30. Depoix, J.; Altmeyer, P. Detecting spectre attacks by identifying cache side-channel attacks using machine learning. *Adv. Microkernel Oper. Syst.* **2018**, 75–86.
31. Zhang, T.; Zhang, Y.; Lee, R.B. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*; Springer: Cham, Switzerland, 2016; pp. 118–140.
32. Zheng, B.; Gu, J.; Wang, J.; Weng, C. CBA-Detector: A Self-Feedback Detector against Cache-Based Attacks. *IEEE Trans. Dependable Secur. Comput.* **2021**. [CrossRef]
33. Bazm, M.M.; Sautereau, T.; Lacoste, M.; Sudholt, M.; Menaud, J.M. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In Proceedings of the 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC), Barcelona, Spain, 23–26 April 2018; pp. 7–12.
34. Cho, J.; Kim, T.; Kim, S.; Im, M.; Kim, T.; Shin, Y. Real-Time Detection for Cache Side Channel Attack using Performance Counter Monitor. *Appl. Sci.* **2020**, *10*, 984. [CrossRef]
35. Kulah, Y.; Dincer, B.; Yilmaz, C.; Savas, E. SpyDetector: An approach for detecting side-channel attacks at runtime. *Int. J. Inf. Secur.* **2018**, *4*, 393–422. [CrossRef]
36. Briongos, S.; Irazoqui, G.; Malagón, P.; Eisenbarth, T. Cacheshield: Detecting cache attacks through self-observation. In Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, 19–21 March 2018; pp. 224–235.
37. Mushtaq, M.; Bricq, J.; Bhatti, M. K.; Akram, A.; Lapotre, V.; Gogniat, G.; Benoit, P. WHISPER: A tool for run-time detection of side-channel attacks. *IEEE Access* **2020**, *8*, 83871–83900. [CrossRef]
38. Mushtaq, M.; Akram, A.; Bhatti, M.K.; Chaudhry, M.; Lapotre, V.; Gogniat, G. Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters. In Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, Los Angeles, CA, USA, 2 June 2018; pp. 1–8.
39. Wang, H.; Sayadi, H.; Rafatirad, S.; Sasan, A.; Homayoun, H. Scarf: Detecting side-channel attacks at real-time using low-level hardware features. In Proceedings of the 2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS), Napoli, Italy, 13–15 July 2020; pp. 1–6.

40. Kim, H.; Hahn, C.; Hur, J. Real-time Detection of Cache Side-channel Attack Using Non-cache Hardware Events. In Proceedings of the 2021 International Conference on Information Networking (ICOIN), Jeju Island, Korea, 13–16 January 2021; pp. 28–31.
41. Bienia, C.; Kumar, S.; Singh, J.P.; Li, K. The PARSEC benchmark suite: Characterization and architectural implications. In Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, Toronto, ON, Canada, 25–29 October 2008; pp. 72–81.
42. Page, E.S. Continuous inspection schemes. *Biometrika* **1954**, *41*, 100–115. [CrossRef]
43. Aminikhanghahi, S.; Cook, D.J. A survey of methods for time series change point detection. *Knowl. Inf. Syst.* **2017**, *51*, 339–367. [CrossRef] [PubMed]
44. Spectre Attack Example. Available online: https://github.com/Eugnis/spectre-attack (accessed on 1 November 2021).
45. RIDL. Available online: https://github.com/vusec/ridl (accessed on 1 November 2021).
46. Flush+Flush. Available online: https://github.com/IAIK/flush_flush (accessed on 1 November 2021).
47. Mastik: A Micro-Architectural Side-Channel Toolkit. Available online: https://cs.adelaide.edu.au/~yval/Mastik/ (accessed on 1 November 2021).