Adaptive Caches as a Defense Mechanism Against Cache Side-Channel Attacks

Sahan Bandara and Michel A. Kinsy
Adaptive and Secure Computing Systems (ASCS) Laboratory
Department of Electrical and Computer Engineering, Boston University
{sahanb,mkinsy}@bu.edu

ABSTRACT

Side-channel attacks exploit architectural features of computing systems and algorithmic properties of applications executing on these systems to steal sensitive information. Cache side-channel attacks are more powerful and practical compared to other classes of side-channel attacks due to several factors, such as the ability to be mounted without physical access to the system. Some secure cache architectures have been proposed to counter side-channel attacks. However, they all incur significant performance overheads. This work explores the viability of using adaptive caches, which are conventionally used as a performance-oriented architectural feature, as a defense mechanism against cache side-channel attacks. We conduct an empirical analysis, starting from establishing a baseline for the attacker's ability to infer information regarding the memory accesses of the victim process when there is no active defense mechanism in place and the attacker is fully aware of all the cache parameters. Then, we analyze the effectiveness of the attack without complete knowledge of the cache configuration. Finally, based on the insight that the success of the attack is heavily dependent on knowledge of the cache configuration, we implement the run-time cache reconfigurations and observe their effect on the success of the attack. We observe that reconfiguring different cache parameters during a side-channel attack reduces the accuracy of the attack in detecting cache sets accessed by the victim by 44% on average, with a maximum of 90% reduction.

CCS CONCEPTS

• **Security and privacy** → **Side-channel analysis and countermeasures**; • **Computer systems organization** → *Reconfigurable computing*; • **Hardware** → Reconfigurable logic applications.

KEYWORDS

Cache side-channel attack, attack mitigation, reconfigurable cache.

ACM Reference Format:

Sahan Bandara and Michel A. Kinsy. 2019. Adaptive Caches as a Defense Mechanism Against Cache Side-Channel Attacks. In *3rd Attacks and Solutions in Hardware Security Workshop (ASHES'19), November 15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3338508.3359574

ASHES'19, November 15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6839-1/19/11…$15.00
https://doi.org/10.1145/3338508.3359574

1 INTRODUCTION

Most modern processors use caches to overcome the "memory wall", which is the widening gap between processor speed and memory access speed. While being vital to the performance of a processor, they have also been the target of numerous side-channel attacks. Several recent side-channel attacks exploit processor caches as the medium that transfers sensitive information from the victim to the attacker [13] [14] [4]. Many of the attacks only use the caches as the covert channel while targeting a different architecture feature, such as speculative execution or out-of-order execution as the point of attack.

Caches are very effective as side-channels, mainly because they are a shared resource and can be used to share information between processes. Primary caches are shared by all the processes running on a core. The Last Level Cache (LLC) is shared among processes from different cores. Other factors that make caches effective side-channels are their high bandwidth, size, and central location in a processor. An important feature of a cache side-channel attack is that it can be software based, and the attacker does not require physical access to the target computer system. Cache side-channel attacks are capable of circumventing security measures, such as privilege checks, address space layout randomization [12] [9], etc. Previous work has shown the viability of same core, cross-core, and cross-VM attacks [29].

Both hardware- and software-based defenses against cache side-channel attacks have been explored [11]. The main drawback of the majority of these defenses is their negative impact on performance. Software-based defenses rely on monitoring the memory access patterns and detecting suspicious processes. Since monitoring and detection use processor time that could have been used to perform useful computations, performance degradation is imminent. After detection, the system may choose to terminate the suspicious process. This gives rise to another issue. If the detection was of low accuracy, there is a risk of the system terminating a benign process. Rewriting applications to remove side-channel vulnerabilities is another approach. However, these are specific to a given application and may still result in performance degradations.

Several cache architectures have been proposed to defend against cache side-channel attacks. However, these designs tend to provide security at the expense of performance, area, and power. Hardware-based cache side-channel defenses depend mainly on isolation and obfuscation. Isolation fragments the cache and limits the cache capacity available to a given process. Obfuscation-based approaches introduce added latency to cache accesses due to the additional logic. It remains an open research challenge to develop a defense mechanism against cache side-channels attacks that (i) does not impose an unacceptable performance penalty on the system and (ii)

provides a means of mitigating the attacks without terminating the suspicious processes. Apart from the above characteristics, the proposed solution can work hand in hand with some of the previously proposed attack detection mechanisms.

The main contributions of this work are:

- An empirical analysis of the impact of run-time cache reconfigurations on the effectiveness of cache side-channel attacks.
- Demonstrating the effectiveness of adaptive caches as a defense mechanism using representative implementations of side-channel attacks and cycle-accurate RTL simulations.
- Demonstrating that even a single run-time cache reconfiguration is sufficient to significantly reduce the effectiveness of a side-channel attack.
- Proposing cache reconfigurations as an alternative response to terminating a potentially malicious process detected by a detection mechanism.

The rest of this paper is organized as follows. Section 2 describes prior work and defines the threat model. Section 3 introduces the proposed defense against cache side-channel attacks. Section 4 presents the results and evaluates the effectiveness of the proposed method. Section 5 proposes cache reconfigurations as an alternative to terminating potentially malicious processes attempting to mount a cache side-channel attack. Section 6 concludes the paper.

2 BACKGROUND AND RELATED WORK

2.1 Cache Side-channel Attacks

Cache side-channel attacks can be used for many purposes, such as stealing cryptographic keys [19], spying on keyboards/mice, breaking kernel address space layout randomization [12] [9], and violating browser sandboxing [18], among others. Cache side-channel attacks exploit intrinsic characteristics of cache systems, such as cache lines mapping to sets, inclusive property, hit/miss time difference, and cache coherence. The timing difference between a cache hit and a miss is the most commonly exploited characteristic. Caches are used to hide memory latency by keeping a small subset of recently used memory contents on the processor die itself, thereby avoiding accessing slower main memory on every memory access. In case of a cache miss, the requested data must be brought to the cache from lower levels of the cache hierarchy, main memory, or disk. This makes the time to service a memory request significantly longer compared to a cache hit, where the requested data is available in the cache and can be sent to the processor immediately. This timing difference can reveal certain information regarding the contents of the cache and the memory access patterns of the processes sharing the cache.

There exist several attack models, such as 'flush+reload' [25], and 'prime+probe' [15]. Although the mechanics may differ, all cache side-channel attacks are based on the same principle: the attacker and victim processes share cache resources, and the attacker can observe or manipulate those shared resources. The cache acts as a side-channel that the attacker can use to monitor the victim process' activity. This is done by monitoring the effects of the victim's activity on the shared cache resources. The general flow of a cache side-channel attack is (i) set the cache contents to a known state, (ii) allow the victim program to run, and (iii) check the state of the cache and try to deduce information regarding the victim's access pattern from the current state of the cache contents. Most of the early cache side-channel attacks depended on the co-location of the attacker and the victim processes. That is, the two processes were executed on the same core. However, this is not common with high core counts in modern processors and cloud service providers actively preventing process co-location in cloud environments. To counter these trends, the recent attacks focus on the Last Level Cache (LLC) to mount cross-core and cross-VM attacks. The authors of [16] & [25] describe such attacks on the LLC.

2.2 Defenses Proposed in Prior Works

Prior works propose two classes of defenses. The first is the set of software-based defenses. These include solutions ranging from application-level answers, such as rewriting applications to reduce information leakage, to system-level fixes, such as monitoring hardware performance counters and using machine learning to predict potential attacks and terminate suspicious processes [5].

Cryptographic applications are the main target of cache side-channel attacks. The vulnerability of these applications arises from memory access patterns that are correlated with the secret information that the attacker attempts to retrieve. For instance, Advanced Encryption Standard AES [6] has been shown to be vulnerable to cache side-channel attacks [19] [3]. While AES can theoretically be implemented completely with arithmetic and logical operations in the processor, some implementations use lookup tables to store the outputs for every possible input for certain steps of the algorithm. Accesses to these tables depend on the secret key and the plaintext to be encrypted. Since all memory accesses go through the cache hierarchy, this correlation between secret data and memory accesses creates a cache side-channel. RSA [21] was also shown to be vulnerable to cache side-channel attacks [20] [26]. The RSA algorithm requires modular exponentiation operations. Certain RSA implementations use a sliding window exponentiation algorithm, which precomputes some values used throughout the exponentiation. Accessing these precomputed values opens the door to cache side-channel attacks to retrieve secret keys by exploiting the correlation between the memory access pattern and the secret key.

The first type of defenses is rewriting the applications in a way that minimizes information leakage. Since this typically prevents usage of performance-oriented techniques, such as precomputing certain values, the resulting implementation will generally perform worse than the original implementation. The other drawback of this approach is that the defense is very application-specific. Rewriting certain legacy applications may not be feasible. A more generic solution that provides a defense for a range of applications is desirable compared to an application-specific solution.

Cache side-channel attacks depend on the timing difference between cache hits and misses. To measure these differences, an attacker needs access to high-resolution timers accessible through the operating system. One proposed defense is to restrict access to high-resolution timers. However, recent works have shown that these attacks can be carried out even without access to high-resolution timers [17]. Another defense, targeting 'flush+reload' and 'flush+flush,' [10] attacks is restricting access to cache flush instructions. An obvious drawback of both these approaches is the fact that benign applications will also be denied access to these

features and it will impact their performance and correct execution. In the case of restricting cache flush, this is a change to the Instruction Set Architecture (ISA) that requires recompiling and, in certain cases, even rewriting an application. Again, this may not be an option with certain legacy applications.

The core challenge faced by software-based defenses is that the side-channel attacks are not based on any particular weakness of the software or the algorithms, but rather on features of the hardware on which the software is executed. Therefore, there is no straight forward defense against side-channel attacks at the software level. While software-based defenses could successfully defend against very specific attacks, those cannot provide any general guarantees about a whole class of attacks or several different threat models. This is due to the specific nature of the software-based solutions, and the lack of control the software has over the underlying hardware. In contrast, if the defenses against side-channels were implemented in hardware, they can prevent attacks on more than one application and will prevent the need to recompile/rewrite every application with security concerns in mind.

The second class of side-channel defenses is hardware-based. We only describe defenses related to cache architecture here. Wang and Lee in [23] describe the Partition-Locked cache (PLcache) and Random Permutation cache (RPcache). The PLcache allows a process to lock cache lines to prevent other processes from evicting it. This essentially creates a private partition inside the cache. PLcache requires the ISA to be extended with a new set of load/store instructions with a lock/unlock sub-operation. The RPcache attempts to randomize the memory to cache mapping by adding another level of indirection in addressing cache sets. A 'Permutation Table' (PT) stores an alternate mapping of the index bits to cache sets. The secure process will use the PT, while other processes will not. This randomizes the set mapping and limits the attacker's ability to infer useful information about the cache sets accessed by the victim process.

Authors of [24] propose an architecture similar to RPcache with a dynamic mapping from memory to cache. They propose a direct mapped implementation with a longer index than that which corresponds to the number of cache lines available. The 'remapping table' and the replacement policy map a larger logical cache to the physical cache while retaining the most recently used lines in the cache. The authors of [8] propose Non-Monopolizable caches. They restrict the maximum number of ways in a set a given process can use. This reduces the ability of the attacker to evict the victim's data from a set and the amount of useful data the attacker can infer. Moreover, this approach does not require extensions to the ISA or support from the operating system. However, it requires additional hardware to keep track of processes using the ways of each set. This could also have a performance impact on benign processes by restricting the effective amount of cache memory available.

Dai and Adegbija in [7] explore the use of reconfigurable caches as a defense mechanism against cache side-channel attacks. They utilize a highly configurable cache, similar to the architecture described in [28], which allows total cache size, line width, and associativity to be dynamically configured. The authors predetermine a set of cache configurations, targeting a particular application or an application domain, which satisfy two conditions (i) selected configurations provide sufficient variability among themselves rendering the information gathered by the attacker less effective and (ii) average energy consumption of the set of configurations is below a predefined energy threshold. A hardware cache tuner applies the set of reconfigurations periodically at run-time. The reconfiguration period is static and determined based on a known attack model. Since a predetermined set of configurations is applied in a predetermined order, it may still be possible for an attacker to design an attack that overcomes the limitations introduced by this method. Having a predetermined reconfiguration window also reduces the effectiveness of this method, since that allows the attacker to modify a given attack to fit the window of time available. The reconfiguration takes place regardless of any other run-time factor. Therefore, even if there is no attack taking place, the cache will reconfigure itself periodically. This will result in performance degradation, as some of the cache configurations may not be optimal for the current application. The authors do not demonstrate the effectiveness of reconfigurable caches as a defense by running concrete implementations of side-channel attacks. Instead, they collect the cache access pattern traces for a non-reconfigurable cache and a configurable cache that reconfigures periodically and calculate the correlation between the two traces to demonstrate the amount of noise introduced by the proposed defense to the traces collected by an attacker.

In summary, previous works include two different approaches to hardware-based defenses against side-channel attacks. The first is isolation, which limits the ability of one process to manipulate the data belonging to other processes. Cache line locking and preventing one process from monopolizing the cache fall under this category. The second approach is obscuring certain information regarding the cache structure and introducing noise to the information collected by the attacker. This approach is based on the fact that cache side-channel attacks require intimate knowledge of the exact cache structure. Without an understanding of the cache parameters, such as cache size, associativity, block size, replacement policy, and cache line mapping function, etc., it is extremely difficult to mount a successful cache side-channel attack. 'RPcache' described in [23] and the defense proposed in [7] fall under this category.

2.3 Performance Impact of Prior Defenses

Different defenses against cache side-channel attacks have varying levels of impact on performance. Continuous monitoring in software to detect side-channel attacks has a high impact on performance, since the monitoring takes CPU cycles from the user applications running on the system. Rewriting applications to limit information leakage typically results in performance degradations, either because of dummy calculations performed to obscure information or due to avoiding performance-improving alternatives, such as using pre-computed lookup tables in computations. Methods based on attack detection usually terminate suspicious processes. If the detection was not accurate and a benign application was flagged, such defense mechanisms can affect not only the performance but also the execution correctness of the application.

Partition locked cache [23], and 'Non-Monopolizable' caches [8] restrict the maximum quantity of cache resources that can be utilized by a given application. This could result in performance degradation for certain applications. The cache architectures described

in [24] and the 'RPcache' [23] dynamically change the address mapping to cache sets. The additional functionality will have an area overhead and an impact on the operating frequency. The reconfigurable cache based approach in [7] applies cache reconfigurations periodically, without paying any attention to currently executing application or whether there is an attack taking place. Since certain cache configurations may not be optimal for all applications, and because reconfiguration also takes time, performance will be impacted by this approach.

2.4 Threat Model

In this work, we assume the following threat model.

- The attacker can execute arbitrary user space programs.
- A process cannot access memory regions allocated for other processes.
- The attacker is capable of mounting a cache side-channel attack on either the primary or secondary caches.
- The attacker is fully aware of the configuration of the caches. (Size, associativity, line width, replacement policy, etc.).
- The attacker does not have access to cache flush instructions provided by the ISA.

3 ADAPTIVE CACHES AS A DEFENSE MECHANISM

Since the success of cache side-channel attacks depends on the understanding of the actual structure of the cache, a dynamic cache structure makes mounting an attack more challenging. This is the rationale behind using run-time reconfigurable caches as a defense against side-channels. We hypothesize that the dynamic nature of the cache will introduce sufficient noise to the data collected by the attacker to prevent accurate information from being retrieved.

Cache side-channel attacks are performed by initializing the cache contents to a known state and monitoring the effect of victim's activity through the changes in the cache contents. Attack models, such as 'flush+reload' and 'flush+flush,' utilize the cache flush instructions provided in certain ISAs to manipulate the cache. Other attack models make multiple accesses to a set of addresses that maps to the same cache set. A set of addresses which map to the same set of a cache is called an 'eviction set' [22]. The success of side-channel attacks depends on the attacker's ability to find eviction sets accurately and efficiently. To find an eviction set, the attacker should be aware of the number of sets in the cache, the size of cache blocks, and the associativity of the cache. An attacker should also know the replacement policy and how the cache sets are indexed. Lower level caches are usually physically indexed and physically tagged (PIPT), while primary caches are virtually indexed to avoid address translation latency. Primary caches can be virtually or physically tagged. If the attacker is to mount an attack on a secondary cache, an understanding of virtual to physical address translation is also required. If an attacker fails to identify an eviction set, the attack is unlikely to succeed. Dynamic reconfiguration of the cache parameters can hinder the ability of the attacker to identify correct eviction sets and mount an attack.

It is possible to declare an array larger than the total size of the cache and use that to prime the whole cache. However, this is not an efficient attack model. It is not necessary to prime every set in the cache at once to mount an attack. Also, accessing the array elements in order does not allow the attacker to prime the cache sets one by one. Instead, way zero gets filled first. Then ways 1, 2, and so on get filled. Therefore, to properly prime even a single set, a large number of accesses are required, as opposed to utilizing an eviction set identified before launching the attack.

Consider a 'prime and probe' attack on a W-way set associative cache using an LRU replacement policy. In the prime step, the attacker primes the cache by accessing addresses known to map to certain cache sets. To prime a cache set with 'W' ways, the attacker makes at least W accesses to the same cache set, in order to evict the current contents of the set and fill it with known values. To do this, the attacker first must find an eviction set of W addresses. Now, assume that the cache is reconfigurable at run-time, and it contains logic to detect the high number of evictions for the same set. The cache control logic interprets the evictions as a result of insufficient associativity and reconfigures to have associativity of $2{\cdot}W$. At this point, the eviction set of W words is useless, as it only evicts half the words in the set. Since the cache has fixed resources, doubling the associativity would mean reducing the number of sets and/or the block size. Changing the number of sets or cache block size changes the mapping from the memory to the cache. With the modified line mapping, some of the addresses in the eviction set may not map to the same set anymore. This makes the original eviction set even more ineffective and, therefore, renders the attack unsuccessful.

If the reconfigurable cache has a static reconfiguration period and if the attacker is aware of it, then the attacker could modify the attack to fit within the time window between two reconfigurations. Although the time constraint limits the amount of information retrieved, the information collected will be accurate. The attacker can still mount a successful attack, albeit one requiring more time and effort. The reconfiguration process takes a certain number of cycles that could have been used for useful computations. The performance impact of reconfiguration increases with the frequency of reconfigurations. Therefore, the frequency of reconfiguration should remain low, which, in turn, makes the time window for the attacker to mount an attack longer. An ideal solution will reconfigure the cache only when a potential attack is detected. At least the reconfigurations should take place at random time intervals.

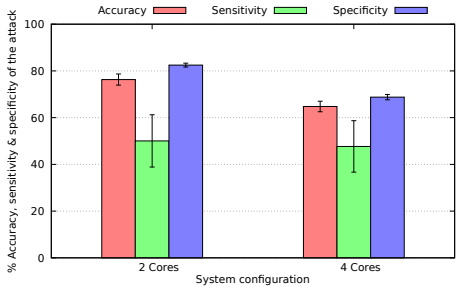

Figure 1: Attack efficacy when targeting the level 2 cache.

If the reconfigurable cache supports a very low number of possible configurations, an attacker may still be able to work around the reconfiguration and complete the attack. An attacker can detect a cache reconfiguration by the excess delay in the cache servicing requests from the processor. Then the attacker can perform a simple preprocessing step to determine the current configuration

of the cache. Once it is found, the attacker performs the attack until the next reconfiguration occurs. However, determining the cache configuration becomes difficult when the cache has a higher number of possible configurations. The attacker's attempts to determine the current cache configuration will take longer due to the large number of potential cache configurations. This will reduce the amount of information gathered by the attacker between two reconfigurations because more time is spent on determining the cache configuration. Therefore, a reconfigurable cache that supports a higher number of configurations will be more successful in thwarting a cache side-channel attack.

3.1 Performance Impact

The proposed defense incurs a performance penalty due to additional stall cycles during a cache reconfiguration and the time taken by the caches to warm-up after a reconfiguration. The latter can be minimized by designing the reconfigurable cache in such a way that the cache does not have to be cleared before a reconfiguration. However, the cache used in this work requires all dirty lines to be written back before a reconfiguration. The cache is empty post-reconfiguration. The performance overhead of this defense mechanism is lower compared to the solutions proposed in previous work. This is because the performance penalty of cache reconfiguration is incurred only when a reconfiguration is performed. In contrast, partitioning the caches and adding more levels of indirection to cache indexing incurs performance penalties constantly, regardless of whether there is an attack taking place. When there are no side-channel attacks detected, the reconfigurable caches can be used as a performance enhancing feature by setting the cache configuration to best fit the memory access patterns of the applications running on the system. Therefore, adaptive caches have the potential to outperform prior solutions in terms of minimizing performance impact.

4 RESULTS AND EVALUATION

4.1 Experimental Setup

The success of a side-channel attack is determined by the amount of information retrieved through the attack. Therefore, the success of a defense mechanism can be measured through the reduction in the amount of information retrieved when the defense mechanism is in place compared to when there is no defense mechanism. We apply the same principle to evaluate the effectiveness of the proposed defense mechanism. First, a simplified model of a cache side-channel attack is implemented and simulated on a RISC-V core adopted from the BRISC-V platform [1]. The simulation is based on two programs. The first is the victim program, which accesses a set of random addresses. The victim program will access 'N' addresses such that if every address maps to a different cache set, 10% of the total cache sets will be accessed. The second is the attack program, which attempts to correctly infer the cache lines accessed by the victim program. In a real side-channel attack, the sensitive information is retrieved by an attacker due to the attacker's ability to detect the cache sets accessed by the victim program. Therefore, measuring the attacker program's ability to correctly detect cache sets accessed by the victim and how the aforementioned ability is impeded by a defense mechanism is a representative measure of the effectiveness of the defense mechanism. The attacker's success is measured by the accuracy of classifying cache sets as accessed by the victim or not accessed by the victim. Each scenario was simulated 100 times with different address sequences accessed by the victim. The attacker takes a simple 'prime+probe' approach in this attack.

We present the accuracy, sensitivity, and specificity of the attacker in detecting cache sets accessed by the victim process. Sensitivity is given by the equation "$true\ positives/(true\ positives + false\ negatives)$". This represents the attacker's ability to correctly flag cache sets accessed by the victim as accessed. Specificity is given by "$true\ negatives/(true\ negatives + false\ positives)$". It shows the ability of the attacker to correctly filter out cache lines not accessed by the victim. Accuracy is given by "$(true\ positives + true\ negatives)/(true\ positives + false\ positives + true\ negatives + false\ negatives)$".

We have simulated side-channel attacks targeting level 1 and level 2 caches. Two different scenarios are analyzed for the case where the two processes are co-located on the same core and the attack targets the level 1 cache. The first is a hardware multi-threaded core where the two processes run on separate hardware threads. The second scenario is where the core only supports a single thread, and the processes are scheduled to favor the attacker. It represents the best-case scenario for the attacker, where they get full control over the cache right before and right after the victim process executes. Therefore, the attack program can prime the cache with no interference from other processes before the victim starts executing. After the victim has completed execution, the attacker gets to probe the cache without any interference from other processes. Since attacker and victim are the only processes scheduled to the core, there is no cache pollution by other processes.

Figure 1 shows the attacker's efficacy when the attacker and victim processes running on two separate cores on a multi-core system and the attacker is targeting the shared L2 cache. The first scenario is a dual-core system where attacker and victim are the only processes running on the system. Accuracy of the attack is lower compared to the attack on L1 cache. This is because the L2 cache is shared between data and instructions. The second scenario is a quad-core system, with two other benign processes running on the system apart from the attacker and victim. Since the L2 cache is shared between more processes, there is higher cache pollution, and the accuracy of the attack is lower.

Among the different scenarios simulated, the one where the attacker and victim processes are manually scheduled one after the other results in the highest success for the attacker. Attacker success is expected because this is an artificial best-case scenario for the attacker. In reality, there is no guarantee that a scheduler will never preempt these processes in the middle of execution. And it is also unrealistic to assume that the only processes running are the attacker and the victim. However, since this experiment is carried out to analyze the impact of run-time reconfigurations on the accuracy of the information collected by the attacker, the best-case scenario for the attacker is used as the baseline for the analysis. After establishing a baseline for attacker success, the level 1 caches in the BRISC-V system were replaced by the run-time reconfigurable caches described in section 4.2 and the impact of run-time cache reconfigurations on attacker success was observed. All simulations are cycle-accurate RTL simulations using Verilog implementations

of a RISC-V processor system and run-time reconfigurable caches. Simulations are run using Mentor Graphics® ModelSim.

4.2 Reconfigurable Cache Architecture

The aim of this work is to analyze the effectiveness of run-time cache reconfigurations against cache side-channel attacks, rather than one particular cache architecture. Although we use a specific RTL implementation of a reconfigurable cache in this work, any other reconfigurable cache architecture should provide similar security in terms of impeding the success of cache side-channel attacks. A highly reconfigurable cache architecture capable of changing its configuration at a fine granularity at run-time is used in this work. The cache architecture does not employ partial reconfiguration techniques, which are specific to Field Programmable Gate Arrays (FPGA). It is built around small independent memory blocks, which can be used as tag or data storage depending on the current configuration and achieves run-time reconfigurability by changing the logical organization of the memory blocks.

The reconfigurable cache architecture is based on the caches available in the BRISC-V platform [1]. It provides the ability to change the associativity, cache line width, and the number of cache sets at run-time. Since the cache line width can change at run-time, modifications are made to the shared bus between L1 and L2 caches, and the bus interface in each cache. These modifications are necessary to properly handle fetching and writing back cache lines and to maintain cache coherence in a multi-core setup.

Another challenge faced when using a run-time reconfigurable cache is maintaining data consistency across reconfigurations. The cache architecture used in this work performs a complete cache cleanup before a reconfiguration. All the dirty cache lines are written back to the lower cache levels. After a reconfiguration, the cache is empty, and this results in low hit rates temporarily. However, this is not a feature required for the proposed defense mechanism. Even if a cache maintains all of its contents across reconfigurations, the run-time changes in the cache structure will reduce the amount of information gathered by an attacker.

The cache structure is further modified to provide a higher degree of flexibility in terms of how the reconfiguration sequence is triggered. A cache reconfiguration can be triggered by writing to a set of memory-mapped registers. This capability also enables programmable software-based methods to be used in reconfiguring the cache. In other words, attack detection algorithms and associated cache reconfiguration parameters can be implemented as software routines or hardware logic. The modified cache structure can interface with either modality (software or hardware) of detection and reconfiguration since it has an internal module that can act as a pass-through or active component. Due to the modular design, new detection schemes can be easily implemented by replacing the 'detector' module. If a software-based detection method is used, the internal trigger signals can be grounded. Figure 2 shows the architecture of the reconfigurable level 1 cache.

The reconfigurable cache architecture used in this work has some key modifications. The main changes related to run-time reconfiguration are concentrated in the 'cache_memory' module, which instantiates the dual-port SRAM blocks that store the cache contents and meta-data. Although, we mainly analyze the effectiveness of cache reconfigurations against side-channel attacks on the level 1 cache in this work, the same defense can be easily adapted to other cache levels.

Table 1: Overheads for the reconfigurable cache architecture compared to the non-reconfigurable one.

# Memory blocks	Unique configurations	Area	Fmax
20	26	+162%	-23%
40	45	+407%	-38%
80	71	+858%	-55%

For completeness of this work, we provide area overhead and reduction in operating frequency for three design points of the reconfigurable cache architecture used in this research compared to the baseline cache design in Table 1. The cache architecture allows an architect to perform a tradeoff between resource usage and the level of reconfigurability by changing the number of memory blocks and the size of a memory block, while keeping the total memory size constant. Table 1 also provides the number of unique configurations for different levels of resource usage. However, it should be noted that the defense proposed in this work is not specific to a particular cache architecture. Any other cache architecture with run-time reconfiguration capability can also be used in the same manner. Different levels of flexibility in the cache architecture may result in different levels of security. Identifying the number of different configurations required to provide sufficient security requires further analysis.

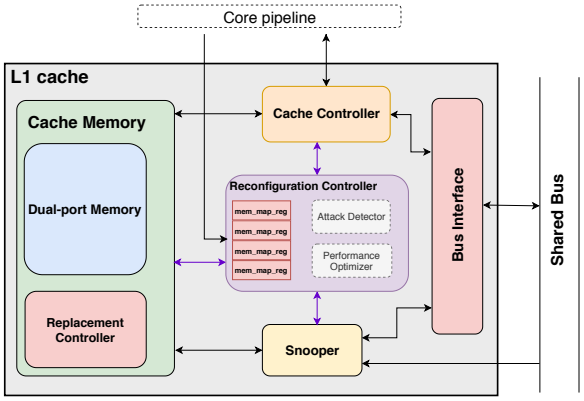

Figure 2: Reconfigurable cache architecture.

4.3 Results

In this experiment, we use the ability of the attack program to correctly identify the cache sets accessed by the victim program as the efficacy/performance metric for the attacker. The motive of the attacker is to correctly classify a higher percentage of sets as accessed or not accessed by the victim. We report the average and standard deviation for the accuracy of the attack program across 100 trials. The plots also show the sensitivity and specificity of the attack program.

Figure 3 shows the attack success under different cache configurations. The attacker is fully aware of the cache configuration and there are no run-time reconfigurations performed in these test scenarios. The baseline configuration considered is a 4-way set associative cache with 64-byte wide cache lines and 128 cache sets. The rest of the cache configurations are derived by changing one

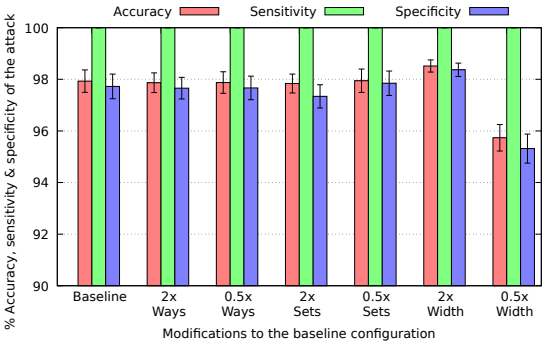

Figure 3: Baseline attack efficacy/performance.

of the parameters from the baseline configuration. The changes include doubling or halving the cache associativity, line width, or the number of cache sets. We observe that the effectiveness of the attack does not vary significantly based on the cache configuration as long as the attacker is fully aware of the exact cache configuration. Accuracy of the attack remains above 95% for all configurations. The attacker can identify all of the cache sets accessed by the victim. Hence the 100% sensitivity. However, the attacker program also incorrectly identifies a few cache sets that were never accessed by the victim program.

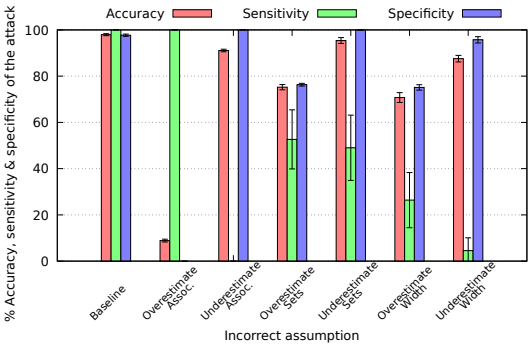

Figure 4: Attack efficacy with incorrect assumptions.

Next, we analyze the effectiveness of the attack when the attacker does not know the correct value of one cache parameter amongst associativity, line width, and number of cache sets. Figure 4 depicts the accuracy of the attack program when the attacker makes an incorrect assumption about one of the cache parameters. We observe that the accuracy and sensitivity of the attack decrease significantly when even one cache parameter is not known.

When the attacker overestimates the associativity of the cache, they construct eviction sets larger than the associativity of the cache. This leads to the attacker evicting part of its known addresses from the cache. Then, the attacker incorrectly identifies the sets that contained the addresses evicted due to their own memory accesses. This increases false positives and reduces the accuracy of the attack. By underestimating the cache associativity, the attacker fails to prime any of the cache sets and ends up identifying none of the sets accessed by the victim, hence zero sensitivity. Similarly, over- and underestimating cache sets result in either the attacker evicting their own addresses from the cache or failing to completely prime the cache. Both result in reducing the accuracy of the attack. Making an incorrect assumption regarding either the cache line width or set count results in an incorrect understanding of address mapping to cache lines. This makes the formulation of eviction sets incorrect and reduces accuracy.

The next step is analyzing the effectiveness of the attack when there are run-time cache reconfigurations. The attack program is designed assuming the baseline cache configuration. We analyze the impact on the accuracy of the attack with cache reconfigurations taking place during different phases of the 'prime+probe' attack. Reconfigurations involving a single cache parameter and multiple cache parameters are analyzed separately. The cases where only one cache parameter changes do not keep the total cache size constant. We allocate sufficient resources at the beginning to accommodate doubling cache associativity, line width, or set count at run-time. In test scenarios where the effective cache size is halved, part of the memory block is not used after the reconfiguration. For the test scenarios where multiple parameters may be reconfigured, the total cache size is kept constant across reconfigurations.

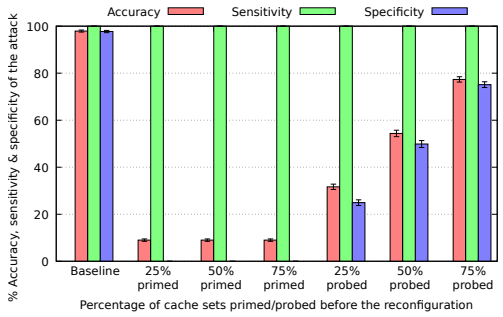

Figure 5: Attack efficacy when associativity is halved.

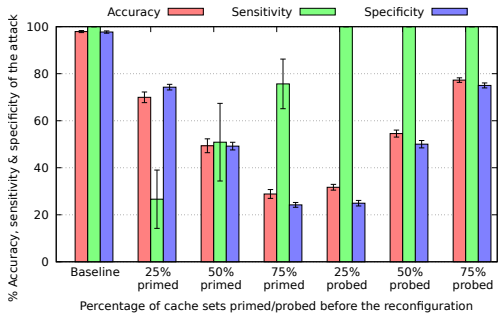

Figure 6: Attack efficacy when associativity is doubled.

Figures 5 and 6 show the impact of doubling and halving the associativity at run-time on the accuracy of the attack. Accuracy increases as the reconfiguration is delayed and the attacker is allowed to complete probing more cache sets. When the reconfiguration is triggered after 75% of the probe phase, attack accuracy is almost 80%. The impact of reconfigurations during the prime phase depends on the exact change made to the cache organization. Reducing the cache associativity results in very low accuracy for the attack. Doubling the associativity shows a lesser impact on the accuracy of the attack. However, the sensitivity of the attack is significantly reduced. Sensitivity increases as more cache sets are primed before the reconfiguration.

Figures 7, 8, 9, and 10 show the results for reconfiguring the number of cache sets and line width during different phases of the attack. We observe similar trends to those seen when reconfiguring cache associativity. Reconfigurations during the prime phase of

the attack decrease the accuracy of the attack by 71% on average. The average reduction in accuracy is 44.4% for the reconfigurations during the probe phase.

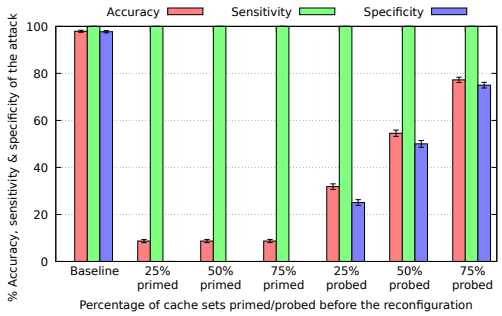

Figure 7: Attack efficacy when cache sets are halved.

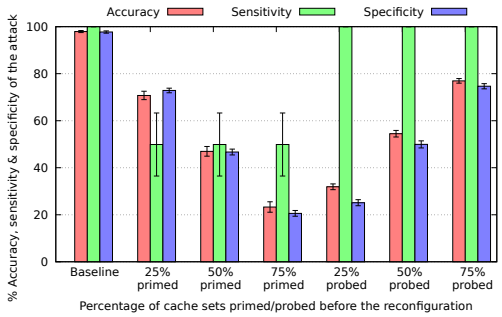

Figure 8: Attack efficacy when sets are doubled.

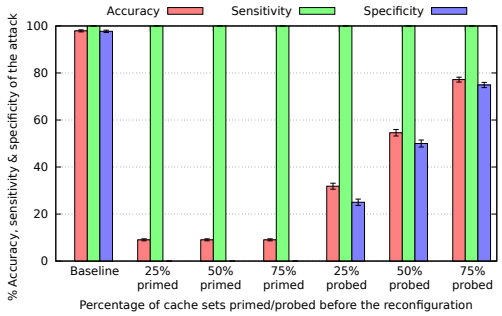

Figure 9: Attack efficacy when line width is halved.

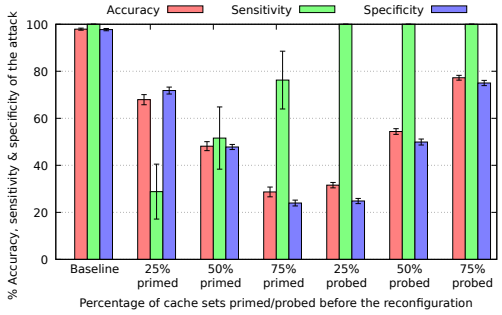

Figure 10: Attack efficacy when width is doubled.

Finally, we analyze the effect of multiple-cache parameters being reconfigured at run-time. This is more realistic because it allows the total cache size to be kept constant, unlike reconfiguring one cache parameter. Figures 11, 12 and 13 show the effect of two cache parameters being reconfigured after 25%, 50% and 75% of the cache sets are primed by the attacker. Similarly, figures 14, 15 and 16 show the effect of reconfiguring two cache parameters at the same time during the probe phase of the attack.

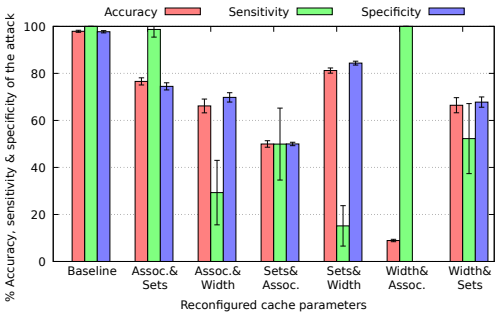

Figure 11: Attack efficacy when two parameters are reconfigured after 25% cache sets are primed.

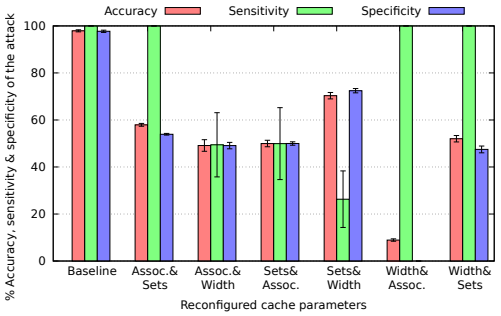

Figure 12: Attack efficacy when two parameters are reconfigured after 50% cache sets are primed.

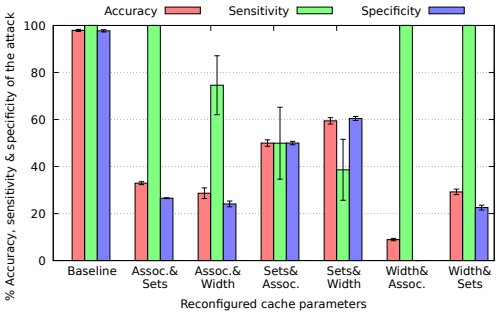

Figure 13: Attack efficacy when two parameters are reconfigured after 75% cache sets are primed.

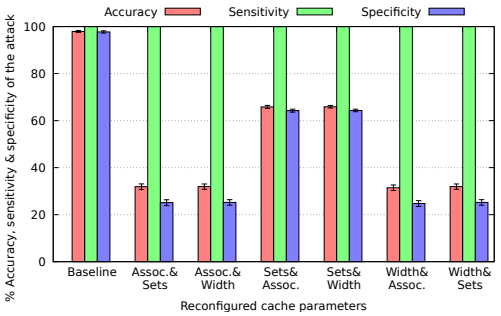

Figure 14: Attack efficacy when two parameters are reconfigured after 25% cache sets are probed.

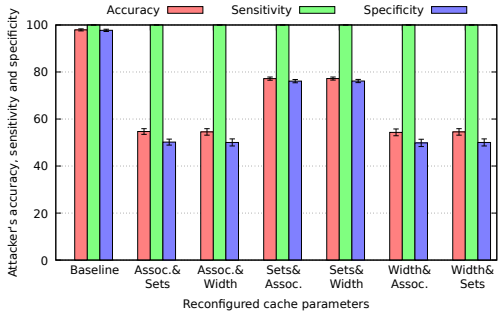

Figure 15: Attack efficacy when two parameters are reconfigured after 50% cache sets are probed.

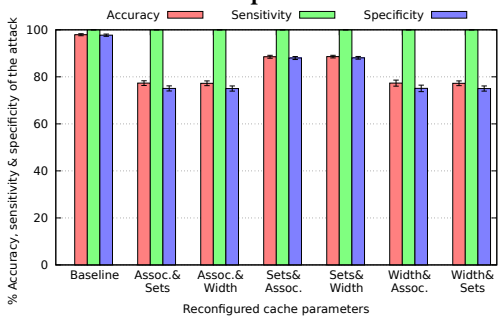

Figure 16: Attack efficacy when two parameters are reconfigured after 75% cache sets are probed.

Similar to reconfiguring one cache parameter, in the cases where the cache is reconfigured during the probe phase of the attack process, the accuracy of the attack improves as more sets can be probed before the reconfiguration. While reconfiguring different parameters results in different levels of impact on the attack performance, the overall trend of increasing accuracy with more sets probed holds. This is intuitive because the attacker is allowed to completely perform the attack on an increasing number of sets in these scenarios. When the reconfiguration takes place after 25% of the cache sets are probed, the accuracy of the attack decreased by 55%. Reconfiguring after 50% and 75% probing reduces accuracy by 36.6% and 17%, respectively.

When the cache is reconfigured during the prime phase of the attack process, the attacker fails to properly prime the cache. This increases the number of false negatives and, therefore, decreases the sensitivity of detecting sets accessed by the victim. Then the probing is done with false assumptions about the cache configuration, which compounds the errors. When the reconfiguration is done after 25%, 50% and 75% of the sets are primed, on average the accuracy of the attack decreases by 40.5%, 51%, and 64%, respectively.

4.4 Evaluation

The test scenarios with no run-time reconfigurations or other programs running on the system yield the highest success for the attacker. This was expected because there is no noise polluting the data collected by the attacker. The simplicity of the attack scenario and the cache hierarchy used in these simulations also result in high success rates for the attack. There are no virtual memory or complex cache line mapping schemes the attacker has to work around. All the caches use physical addresses to map cache lines. The processor is running bare-metal code. Adding other benign processes to the system reduces the success of the attack. This is due to caches now being shared by all the other processes. The attacker cannot distinguish between the accesses from the victim and the other processes not targeted by the attack. This observation is true for both the multi-threaded and multi-core test scenarios we have analyzed.

Run-time reconfigurations drastically reduce the attack's success. Even a change in one cache parameter renders the information gathered by the attacker more or less unusable. This is because the attack is designed assuming a certain cache configuration. Once the cache configuration changes, some of the assumptions made in designing the attack become invalid. Therefore, the information collected by the attacker becomes unreliable. A single reconfiguration was shown to be sufficient to reduce the accuracy of the attack. However, these experiments were conducted using a simple implementation of a cache side-channel attack that does not attempt to identify a cache reconfiguration and the new cache configuration after a reconfiguration. If a sophisticated implementation of the attack was used, multiple reconfigurations may be necessary. A cache with a higher number of possible configurations can counter the attacker's attempts to identify the new cache configuration and then modify the attack to fit the new configuration. When there is a large space of potential configurations, identifying the current cache configuration takes longer and that reduces the time available to mount the actual attack.

Another observation made was the effect of how long the attack continued before a cache reconfiguration takes place. The time taken by the detection mechanism to detect a potential attack and trigger a cache reconfiguration is directly related to the success of the attack. Reconfigurations earlier in probe phase result in a larger reduction in accuracy of the attack. Reconfigurations during prime phase on average result in a higher reduction in accuracy than reconfigurations during the probe phase. This demonstrates the importance of early attack detection.

5 RECONFIGURATION AS AN ALTERNATIVE RESPONSE

In this work, we assumed that there is a detection mechanism in place that will detect potential cache side-channel attacks and trigger cache reconfiguration. Future work includes implementing a real-time side-channel attack detection mechanism. Most of the prior work in real-time side-channel attack detection utilizes learning-based methods [5] [2] [27]. While prior work has shown high accuracy in attack detection, false positives are not eliminated by those detection methods. A false positive detection can result in terminating a benign process. The alternative is human intervention when the detection mechanism identifies a potentially malicious process. However, this is also not a practical solution due to the sheer number of processes running on a system. For certain use cases, such as a cloud service provider, most of the processes will be running client code, which the service provider has little to no information on. Terminating a client's benign process can have financial repercussions.

Run-time cache reconfigurations can be used to complement some of the real-time detection methods proposed in previous works. Cache reconfiguration is orthogonal to attack detection, and, therefore, the detection mechanism can be implemented independently. It can be hardware- or software-based. As discussed

earlier, terminating a process is not an acceptable response in most instances of an attack detection system flagging a process as suspicious. Reconfigurable caches provide another viable response to a potential side-channel attack. If the detection mechanism detects potential attacks with varying degrees of confidence, it is important to have a defense that does not include potentially terminating a benign application. In a system with run-time reconfigurable caches, a low confidence detection of a potential attack could trigger a cache reconfiguration. High confidence detections can still terminate the suspicious process. With this scheme in place, even if the detection method has incorrectly flagged a benign process, it will only experience performance degradation, as opposed to termination.

6 CONCLUSION

In this work, we analyzed the viability of adaptive caches as a defense mechanism against cache side-channel attacks. Our observations indicate that the success of a cache side-channel attack is severely impeded by run-time cache reconfigurations. We observed that a single cache reconfiguration can reduce the ability of the attacker to identify the cache sets accessed by the victim drastically. The accuracy of the attack decreased by 57.6% on average when a single cache parameter reconfiguration was performed without maintaining the total cache size. Reconfiguring two parameters while keeping the cache size constant resulted in an average reduction of 44.3% in accuracy. We were also able to demonstrate the importance of early attack detection with our results. Against a 'prime+probe' attack, cache reconfigurations taking place after 75%, 50%, and 25% of the probe phase resulted in decreasing accuracy for the attack. Reconfigurations during the prime phase resulted in even lower success for the attack. We discussed how run-time cache reconfigurations can be used as an alternative response to potential side-channel attacks instead of terminating the suspicious process. This reduces the potential of terminating a benign process due to a false positive from the attack detection mechanism. With a run-time cache reconfiguration, the process will experience performance degradation as opposed to being terminated. The major takeaways of this work are: (i) reconfigurable caches are an effective defense mechanism against cache side-channel attacks; (ii) they provide a strong defense with a negligible performance overhead compared to previously proposed methods; (iii) the speed of detecting a potential attack is paramount to the success of the defense; (iv) using run-time cache reconfigurations to thwart cache side-channel attacks reduces the pressure on attack detection; (v) reconfiguring even a single cache parameter reduces the accuracy of a cache side-channel attack significantly.

REFERENCES

[1] Sahan Bandara, Alan Ehret, Donato Kava, and Michel Kinsy. 2019. BRISC-V: An Open-Source Architecture Design Space Exploration Toolbox. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*. ACM, New York, NY, USA, 306–306. https://doi.org/10.1145/3289602.3293991

[2] Mohammad-Mahdi Bazm, Thibaut Sautereau, Marc Lacoste, Mario Sudholt, and Jean-Marc Menaud. 2018. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In *Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 7–12.

[3] Daniel J Bernstein. 2005. *Cache-timing attacks on AES*. Available at http://cr.yp.to/papers.html#cachetiming.

[4] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 991–1008.

[5] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing* 49 (2016), 1162–1174.

[6] Joan Daemen and Vincent Rijmen. 1999. AES Proposal: Rijndael. Available at https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.

[7] Chenxi Dai and Tosiron Adegbija. 2017. Exploiting configurability as a defense against cache side channel attacks. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 495–500.

[8] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 35.

[9] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 40.

[10] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.

[11] Hossein Hosseinzadeh, Mihailo Isakov, Mostafa Darabi, Ahmad Patooghy, and Michel A. Kinsy. 2017. Janus: An uncertain cache architecture to cope with side channel attacks. *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2017), 827–830.

[12] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical timing side channel attacks against kernel space ASLR. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 191–205.

[13] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).

[14] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).

[15] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 605–622.

[16] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*. 605–622. https://doi.org/10.1109/SP.2015.43

[17] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L Titzer, and Toon Verwaest. 2019. Spectre is here to stay: An analysis of side-channels and speculative execution. *arXiv preprint arXiv:1902.05178* (2019).

[18] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. 2015. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1406–1418.

[19] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Cryptographers' track at the RSA conference*. Springer, 1–20.

[20] Colin Percival. 2005. Cache missing for fun and profit.

[21] R Rivest et al. 1983. Cryptographic communications system and method, US 4405829 A. (1983).

[22] Pepe Vila, Boris Köpf, and José Francisco Morales. 2018. Theory and practice of finding eviction sets. *arXiv preprint arXiv:1810.01497* (2018).

[23] Zhenghong Wang and Ruby B Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, Vol. 35. ACM, 494–505.

[24] Zhenghong Wang and Ruby B Lee. 2008. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 83–93.

[25] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. 719–732.

[26] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering* 7, 2 (2017), 99–112.

[27] S. Yu, X. Gui, and J. Lin. 2013. An approach with two-stage mode to detect cache-based side channel attacks. In *The International Conference on Information Networking 2013 (ICOIN)*. 186–191. https://doi.org/10.1109/ICOIN.2013.6496374

[28] Chuanjun Zhang, Frank Vahid, and Walid Najjar. 2003. A highly configurable cache architecture for embedded systems. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.* IEEE, 136–146.

[29] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 305–316.