The Ghostbusting Brochure: A Systematic Analysis of Real Time Spectre Side-Channels Detection

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Dhruv Pandya

Fall 2023

Tianhao Wang, Department of Computer Science

The Ghostbusting Brochure: A Systematic Analysis of Real Time Spectre Side-Channels Detection

CS4991 Capstone Report, 2023

Dhruv Pandya
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
dhp2rpe@virginia.edu

***Abstract*—Since Spectre, various detection-based Side-Channel Attack mitigations have been proposed. They monitor systems for signs of ongoing SCAs, supplement built-in security mechanisms, and can be critical in detecting and preventing zero-day attacks.**

Existing SCA and malware detection surveys tend to focus on machine learning based classification methods. In this survey, we extend the spotlight to include monitoring systems that provide the data and alarm systems that act on the results. We provide a practical guide for navigating implementational trade-offs by proposing cost metrics for method comparison and measuring sensitivities to various parameters in system design.

I. Introduction

Microarchitectural side-channel attacks are a class of information-stealing attack that exploit discrepancies between a system's physical implementation and logical design. Before 2018, academics and hardware designers believed that side-channel attacks are only a threat to improperly designed cryptographic algorithms and can be easily addressed by constant-time programming and a host of similar techniques. As a consequence, side-channel attacks were studied and monitored, but not treated as a major security concern.

In 2018, the discovery of Spectre and Meltdown attacks shook the hardware security community by demonstrating how the vast complexity underlying modern processors could be used to execute side-channel attacks previously thought impossible. The discovery of Spectre and Meltdown demonstrate the immense complexity of modern CPUs, and the security issues that comes with it. Spectre brought behind-the-scenes optimizations into the spotlight, as well as the enormous CPU flaw exposed by the ability to reach impossible program states via transient execution combined with return oriented programming, threatening data previously considered impossible to exfiltrate.

Type	Description	Advantage	Disadvantage
Patching	Patch software to avoid triggering hardware vulnerabilities.	No hardware changes.	Need to secure all software, can be broken by software updates.
Hardware [27], [31], [36]	Modify hardware design to forestall attacks.	Guaranteed security.	Require hardware changes; adaptions required to new hardware optimizations.
Detection [20] [1] [16]	Monitor execution to detect attacks.	Covers 0day attacks, requires no hardware change.	Reliability and overhead.

TABLE I: Software, passive, or and detection-based side-channel mitigations.

Since then, there has been great interest in transient execution attack mitigations. Mitigations can be categorized as software, passive, or detection-based (see Table I). Software and passive mitigations can cover massive attack surfaces. They have hitherto rightly been treated as the primary method of defending against side-channel vulnerabilities. However, as novel hardware optimizations are introduced and new attacks surface [3], [24], existing mitigations need to be expanded to cover the newly exposed surfaces. Any delay in doing so results in zero-day attacks.

To bridge the gap in defenses, detection-based mitigations such as NightsWatch [16] and PerSpectron [15] have been developed to monitor the execution of workloads and report either anomalies in workload execution (known as Anomaly-based Detection) or execution patterns that match the signature of side-channel attacks (known as Signature-based detection) [29]. Before 2018, a substantial amount of work has already accumulated of execution-signature based malware detection. The threat of Spectre and Meltdown caused much research interest from malware researchers and hardware researchers alike. Older malware detection methods have been adapted for side-channel detection, new methods have been developed, and numerous survey papers exist on the topic [14], [19].

While Spectre-era side-channel defense surveys [19] provide broad and detailed analyses of the statistical and machine learning methodologies for distinguishing between samples taken from malicious and benign processes, they tend to neglect an explanation of the method by which these samples are collected in the first place. From existing surveys, it is unclear how to tackle the tradeoffs involved in the design of the sampling systems.

In this work, we will focus on active detection-based mitigations for Spectre and other attacks, while also drawing from the pre-Spectre body of work on malware detection to inform our evaluation. We present an in-depth, survey of implementational parameters of Spectre detection systems, followed by a comprehensive, systematic literature evaluation from an implementational perspective, with a focus on practical features such as performance overheads, ease of integration to real world systems, false-positive and false-negative rates, quality of cross-validation, and potential to detect zero-day attacks. We also perform experiments to evaluate the performance of side-channel detectors across a wide range of system parameters, including different machine learning methods, the type and number of performance counters tracked, and overhead and accuracy tradeoffs caused by different sampling frequencies and monitoring granularities. We conducted this survey to fill in some gaps and provide more useful guidance for industry practitioners as they choose the best detector for their application.

Our exploration is centered on Spectre detection mechanisms, but we draw from the malware detection and general side-channel detection mechanisms as well. We go in-depth on approaches based on existing hardware performance counters, but we also discuss methods that use instrumentation [25], simulation-based data gathering, and novel hardware features specifically designed for the purpose [15], [35].

II. Background

As a result of their complicated architecture, modern processors are susceptible to a variety of security flaws that hackers exploitto steal information directly. Side Channel Attacks allow hackers go a step farther by indirectly retrieving secret information by monitoring a system's cache. Now a significant problem for high-level PCs and Cloud Machines containing sensitive data, such attacks measure the shared cache memory's access time and use variations thereof to infer sensitive data. This is worsened by speculative execution, an optimization that boosts the machine's performance but results in unintuitive execution behavior and difficult-to-secure caches. During the subsequent speculative execution, a virus or attack could use these leftover memory accesses to gain access to the system's data. The form of attack is a vast topic that encompasses a variety of exploits employing various approaches to discover secret information via CPU side channels [12], [20].

A. Performance Counters

Hardware performance counters are registers with a special function integrated into modern microprocessors. They can be configured to track a number of important computer system metrics for analysis purposes. These metrics may include, clock cycles, cache hits and misses, and branch misses. In our situation, the hardware-based performance counters are mostly used in the occurrence of an event such as side channel attacks. Due to the fact that they are hardware-based in the CPU, they are very quick and incur no performance overhead, making them ideal for real-time detection. Interfaces to access hardware performance counters include `PAPI` (Performance Application Programming Interface) [7] and `linux-perf` [1].

B. Side-Channel Mitigations

Passive mitigations often attempt to minimize the capacity of attackers to (1) cause temporary execution, (2) access sensitive data during transient execution, and/or (3) transmit stolen data to committed execution via patching software and hardware. While these approaches provide security-by-design, they add complexity to the hardware design process since they must be constantly enforced as hardware architecture evolves to prevent innovative attacks that use different areas of the micro-architecture to bypass defenses.

Active mitigations, on the other hand, are typically software-based mitigations that identify the existence of malware or an assault in the cache and inform the user during run-time. They can be configured to detect signatures similar to known assaults or atypical execution patterns that do not match existing workload signatures [29]. This affords them the opportunity to detect previously unidentified side-channel attacks. Active mitigations should not be considered a replacement for security-by-design, but rather as a supplement that can be used to provide early alerts of new attack variants and help hardware designers in implementing the required security measures.

Following the collection of data, a statistical model or a form of machine learning may be utilized to differentiate between attacks and non-attacks. Existing side-channel surveys concentrate extensively on this step, the type of computer utilized, the performance counters measured, and even the experiment's precision. These, however, exclude essential statistics such as performance overhead and granularity implementation. More often than not, articles rehash the same material without adding to the body of knowledge on the subject by performing real-time detection for various cache-involving execution approaches [20].

Active side-channel detection techniques typically require data to be gathered using existing hardware performance counters, but there are proposals using instrumentation, simulation-based data gathering, and occasionally novel hardware features designed specifically to detect side-channel attacks [15], [35]. The collected data is forwarded to a classification algorithm, typically a machine learning model, to recognize certain specific events that occur in a CPU, along with an estimate of the virus's likelihood. Existing side-channel research focuses mostly on methodological considerations and the capacity of various defenses to detect various types of attacks. One of the key strengths of active mitigations is their potential to detect zero-day attacks, which exploit previously undiscovered security flaws.

C. machine learning Based Attack Detection

machine learning is a class of statistical methods that learns to use past fed input data without being told explicitly. They need large amounts of data and usually process a pattern through neural networks. In trying to detect side channel attacks, machine learning can be a very useful technique for targeting attacks by learning of the ways in which it manipulated different parts of the machine. machine learning has already shown it can be used for cybersecurity, with its use in malware detection and post-silicon validation [21]. Currently, ML models generated operate in a black-box fashion. The research done in the machine learning area has therefore gone into creating explainable machine learning models that can create a label with a reason for the output [10]. This can help to incorporate more relevant samples and generate more accurate results. In case of side channel detection mechanisms, the goal would be to figure out all the kind of attacks possible.

D. Cache Side Channel Attacks

1) Prime + Probe: Prime + Probe attacks are a type of cache side channel attack that exploits caching to exfiltrate data access patterns patterns and sensitive data. They can target various levels of the cache hierarchy from LLC [5] to the micro-ops cache [24]. The attack consists of three phases: PRIME, IDLE, and PROBE. During PRIME, A spy evicts cache lines and fills the cache with a portion of its own memory known as an "eviction set." In IDLE, the spy waits for a predetermined amount of time while the target performs sensitive tasks. During the PROBE phase, the spy attempts to retrieve the eviction set. If the eviction set is not in the cache, as evident by loading taking a long time, the spy infers that the victim has visited sensitive memory locations that mapped to the same cache lines as the eviction set.

2) Flush + Reload: Flush + Reload improves upon Prime+Probe by using the `CLFLUSH` instruction to reliable flush victim data from the cache. It has the highest resolution among cache side channel attacks and can even collect keystroke data, but requires the ability to flush the victim's memory from cache. Flush+Flush has only 3 stages: FLUSH and RELOAD. During FLUSH, victim memory is flushed from cache. The attack then goes into the RELOAD phase, where the victim is allowed to execute and is timed. Long reload times indicate that the victim did not access the shared page's sensitive data, allowing the spy to establish the victim's access patterns to sensitive data [5].

E. Speculative Side-Channel Attacks

Together with Meltdown, Spectre [22] was a security flaw found in 2018 that could also steal information from any computer at the time. The Spectre attack employs branch prediction flaws to circumvent user and operating system isolation. There are two primary forms of the spectre exploit: variant 1 and variant 2. The first variant exploits speculative execution to evade memory access boundaries check conditional branch instructions [22]. The problem arises when the process accesses out-of-bounds memory speculatively, prior to the completion of validation checks. During this time frame, speculative memory can leave gaps and leaks that can be exploited by hackers via the generated side channel. For indirect branches, the second type employs speculative execution rather than branch detection. An attacker is able to modify these branch predictors to execute new code and reveal sensitive data [20].

For each variant, Spectre is able to retrieve sensitive information and load them into speculative memory. The information is then leaked to the attacker via another side (covert) channel like Prime+Probe or Flush+Reload.

Spectre is able to bypass existing side-channel mitigations and can show its own unique performance counter motifs (for example, in our exploration, we observe that processes executing the Spectre attack exhibit abnormally low branch miss rates that are not expected for typical side-channel attacks). However, as Spectre requires a traditional Side-Channel to transfer the stolen data from speculative to committed execution, we expect it to exhibit performance counter motifs identical to other side-channel attacks unless it disguises itself by inserting noise or adversarially avoiding machine-learning based detection mechanisms [10], [33].

III. System Parameters and Their Impact

In this section, we present a deep dive into real-time microarchitectural side-channel attack detection methods, and discuss the design choices available and how they affect tradeoffs with performance, accuracy, and ease of implementation. Most of the research surveyed are OS-level mitigations, integrating with `linux-perf` to gather perfomance data for classification into side-channel and non-side-channel categories. We explore the research surveyed along the following axes:

A. Level of Oversight

For side-channel attack detection, the level in which the process can be done varies. In this current taxonomy, every paper uses an OS for Granularity of Oversight. This means that they are either simulations that are run on MARSSx86 emulators or actual machines with intel chips/x86 [4], [30]. Most of these papers offer the same solution, so it is more likely than not that they build on each other's work and experiment on a higher, OS, level of architecture. The PerSpectron paper is the only one that does not follow this [15]. It has a Micro-architectural Proposed granularity of oversight. This helps it to detect attacks before they start, and offers a deeper level of security than at the OS level of detection.

B. Sampling Interval

The literature reports sampling intervals in various units, but most works provide an interval measured in milliseconds (e.g. [21], [1], [11]). Reported units include:

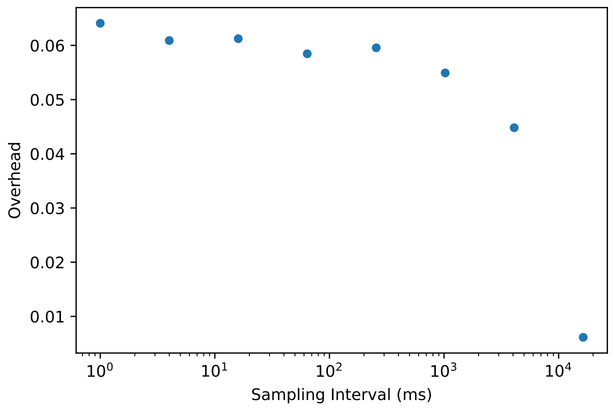

Fig. 1: Overhead decreases with increasing sampling frequency.

- Time (The majority in *ms*, with some reports in *Hz* and *μs*) [4], [28]
- Encryption Rounds (when the victim is AES) [17]
- Cycles (Ranging from 25,000 to 1,000,00) [6], [35]
- A range (for variable-frequency sampling schemes like Cache-Zoomer [8], whose sampling frequency depends on processor clock rate and cache activity)

Sampling frequencies in literature exhibit a wide variation. Typical systems report a time-based sampling frequency, with an interval between 1ms and 1000ms. Frequencies under 1ms are also reported, but rare due to quickly increasing overheads. To control the overhead, The PerSpectron paper [15] proposes extra hardware to take and record performance counter samples without interrupting the flow of execution, and is able to sample at 0.003ms per sample. This is the lowest sampling frequency reported in the literature. Many papers take samples using `linux-perf` and report a sampling frequency of 1ms [2], [34], [37], which is the default sampling frequency in `linux-perf`.

Shorter sampling intervals produce greater overheads, but this comes with two advantages. First, a shorter interval allows the detection mechanism to isolate attack-phases for a clearer signal, and makes them harder to defeat via evasive attacks like Kurvila's Paper [10]. Second, they have a better chance at terminating an attack before it can complete, as the attacker is more likely to be caught in the act of performing the attack. This makes them suitable for fully automated systems that automatically terminates suspicious activity.

C. Detection Methodology (Statistical / ML / Rule-based)

Two historical papers from the CC Hunter line of research classify processor behavior observations into safe and malicious categories by thresholding statistics on a histogram of memory addresses. CC-hunter [4] uses memory autocorrelations to detect periodic cache attacks. ReplayConfusion [35] takes this logic one step further by re-executing the program with different cache mappings and observing the difference in autocorrelation. Notably, they rely on thresholds picked by researchers and do not perform training.

In contrast, most of the papers surveyed [10], [20], [28] use traditional supervised learning methods like Ridge Regression, SVM, and Random Forrest to classify data as safe or malicious.

There are also an abundance of research [5], [32] that use deep learning to detect real-time side-channel attacks. They measure metrics like branch misses, cache levels, and cache misses, and use

ML in software to implement their models. Compared to traditional ML models, they can recognize more complex patterns but are less explainable and can overfit. There are proposals to address this with explainable ML [10], [21].

In our own tests (see Figure 3), we explore some of these tradeoffs. For data gathered through standard performance counter sampling, we find that deep learning was not necessary and that random forests provide much performance at little cost.

D. *Attacks Targeted*

Cache Side Channel Attacks are a large range of attacks involving a variety of antiquated and modern malware. Consequently, the sort of attack has varying effects depending on where it occurs within the system. The papers in this taxonomy provide a variety of malware and attacks: Malware Cache Side Channel Attacks, Covert Timing Channel Attacks, and Spectre Side Channel Attacks. Malware Cache Side Channel Attacks are the subject of articles such as [5], [28], which demonstrate how more information can be accessed from the system or hardware. They explain the many types of malware that induce side-channel assaults and the necessary detection techniques. Covert Timing channel assaults transmit data to a target system by altering its behavior over time. This is used by attackers to tailor the system to their needs and gain access to its private portions. The [4], [35] articles discuss this type of attack as their primary objective, while also touching on malware detection for instances such as Trojan Horse attacks.

Recent research has paid significant attention to Spectre, and [1], [12], [13], [20], [30], [32], [38] demonstrates how real-time detection of Spectre side-channel attacks can be studied. Non-spectre publications appear to concentrate mostly on cache behavior, but spectre-targeting defenses frequently incorporate branch behavior, as branch mistraining is essential for controlling speculative execution. Particularly, Spectre attacks are characterized by extraordinarily high branch miss rates, as mistraining repeatedly teaches the CPU a false, predictable pattern before each mis-speculation attack.

E. *Zero-Day Attack Detection*

One of the key advantages of performance-counter based attack detection is their potential to detect zero-day attacks. This can happen in two ways:

- Signature-based detection, which detects new attacks based on partial similarity to existing attacks
- Anomaly-based detection, which detects new attacks based on difference from known-safe workloads

Signature-based zero-day detection can be useful in recognizing new attacks, but fall short when presented with radically new attacks that have no similarities with existing ones. Examples include [18], [26].

Anomaly-based methods seek to identify both zero-day assaults and existing threats. Using statistical or machine learning approaches, they only analyze normal execution behavior. Attack detection is done instead by defining large divergence from known-good behavior as anomalous and raising alarms. Despite better potential to detect zero-day attacks, there is significantly less research for anomaly-based detection [37], and existing studies report higher false-alarm rates. Additionally, when the execution behavior of the system changes (such as when new workloads are added), anomaly-based methods can fail.

F. *Response to Suspicious Activity*

Once the classification system triggers an alarm, there are a number of ways the system responds to the alarm. In this section, we discuss the different measures taken in the papers surveyed, and outline how well these measures tolerate false alarms.

Works surveyed that do have a proposed alarm mechanism falls under one of the the following categories:

1) The system may terminate the process that is executing the malicious code [1], [11], [16], [20], [28].
2) The system may try to prevent malicious execution non-destructively, such as freezing the suspicious process or executing it in a sandbox to further check whether it is malicious [35].
3) The system may log the incident for later discovery by a human operator.

Each of the proposed countermeasures has its own advantages and disadvantages. Terminating the process is the most aggressive response, but it is also the most effective. It is also the most straightforward to implement, as it does not require any additional infrastructure. However, it is also the most disruptive, may cause the process to lose unsaved work, and can lead to Denial of Service attacks. The false-alarm tolerance depends on the way the system is built to tolerate process terminations.

Logging the attack is much less costly, but depending on the application, can be ineffectual for attack prevention. It does not enable denial-of-service attacks, and can help system administrators to identify and mitigate attacks. However, it requires additional infrastructure to log the attacks, and requires human intervention to take action. While it has some false-alarm tolerance, too many false alarms can make real attacks difficult to detect.

Freezing the process strikes a balance between low disruptiveness and high false-alarm tolerance. However, it can be more difficult to implement, as it requires additional infrastructure to hold the process's state, and can cause rapidly increasing queue lengths if triggered repeatedly in latency-critical systems. However, it is less disruptive, and does not cause data loss. This response is also more tolerant of false alarms, as it does not require the process to be terminated.

IV. Standards for Evaluating Side-Channel Defense Performance

In this section, we present surveys of 70 attack detection methods, and discuss the tradeoffs between them. We also confirm some of their claims with our own experiments.

Most of the research surveyed are OS-level mitigations, integrating with `linux-perf` to gather perfomance data for classification into side-channel and non-side-channel categories.

Performance Overhead in Side-Channel detection mechanisms can be grouped into 2 main categories, Hardware and Timing Overhead.

A. *Sampling Overhead*

A percentage-slowdown overhead was only reported in 17 out of 75 relevant papers. The papers surveyed tend to show minimal execution time increases between 1% to 5% , with one study reporting overheads as high as 15.25% [18]. Other papers mention timing overheads, but do not provide numerical measurements (e.g. [15], [17], [20]). In the majority of the remaining papers surveyed, timing overheads are conspicuously missing. Crucially, very few papers provide any information on how timing overhead is affected by adjusting data gathering parameters such as sampling frequency or granularity of oversight.

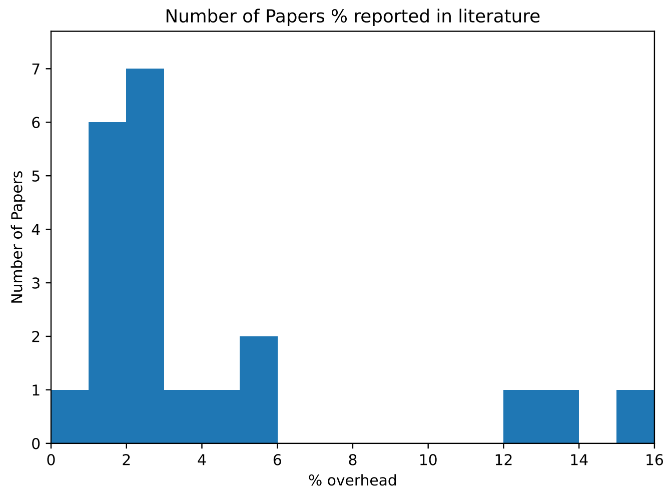

Fig. 2: Existing side-channel detection literature reports low overheads.

Without knowing the overhead, it is impossible to know whether a given detection mechanism will slow workloads down too much for detection to be practical. Without knowing the tradeoffs between accuracy and overhead, it is impossible to know how to tune a detection mechanism to achieve the desired balance between the two.

To give a sense of the trade-off between timing overhead and hardware performance counter sampling frequency, we used the `perf` tool to measure the overhead of sampling the performance counters on a 2.6 GHz Intel Core i7-6700K CPU. We ran `SPEC CPU 2017`. We then ran the same program while sampling the performance counters at different frequencies. The overhead of sampling the performance counters between 1KHz and 1/16HZ ranged between 6.4% and 0.61%.

The results are shown in Figure 1. The figure shows the overhead of the hardware-based detection mechanism as a function of sampling frequency. The overhead is measured as the percentage of time spent in the detection mechanism. The overhead is low at 6ms, and decreases with increasing sampling frequency. This is expected, as perf only actively interrupts the program's execution to record data during the sampling period. The overhead is consistent with the overhead of the software-based detection mechanisms in the literature.

Based on the literature survey and our measurements, software measures for detecting side-channel attacks based on hardware performance counters has acceptable ranges of overhead for most detection frequencies. Higher-overhead approaches tend to use instrumentation and other more invasive methods. Hardware-based methods tend to have overheads that vary greatly, and different researchers report different measures of overhead.

B. Classification Overhead

Many of the papers report overheads only for the sampling mechanism, and not for the classification algorithms. For example, while the PerSpectron [15] reports no numeric overhead, it derives its ”negligible” overhead estimate purely from the sampling mechanism design. Although the classification algorithms may comprise a large portion of the computational complexity of a detection scheme, reporting only the sampling and enforcing overhead can be appropriate since classification overheads can be offloaded to other cores or a co-processor, such that they can run parallel to the workload without slowing it down. Methods for accomplishing this include:

- Using specialized edge ML accelerators to handle classification [9].
- Using compressed sensing techniques to create small but detailed performance monitoring packets that can be sent to remote servers for analysis [34].

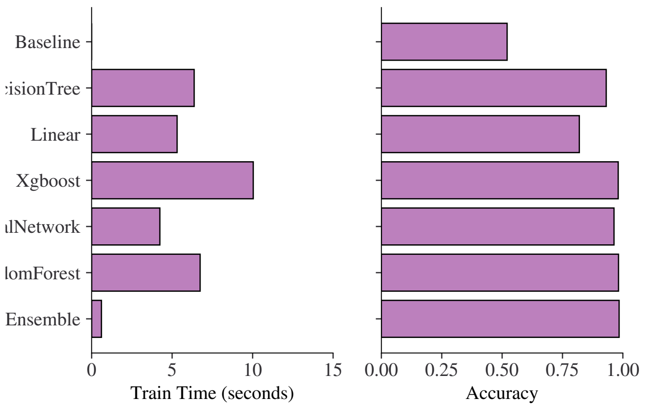

Fig. 3: Accuracy and training time across different models.

In figure 3, we measured time taken to train various statistical models for classifying performance counter data, which provides a baseline for estimating the inference overheads for these models.

C. Hardware Overhead

Hardware overheads are reported by several hardware-based defense papers [6], [10], [30]. Most papers report overheads of less than 5%, and more recent works tend to report overheads more consistently.

A notable exception to the norm of reporting low overheads is ReplayConfusion [35], which claims that there is no comprehensive, low overhead defense for LLC-based covert channels.

In addition to under-reporting overhead like the software-based detection literature, different hardware mechanism papers use different metrics for overhead reporting, including changes in transistor count [30], power draw [10], and hardware area [10].This makes it difficult to compare overheads between the papers that do report them.

One notable work here is Kuruvila (2021), which extensively tested non-interrupting custom HPC monitoring systems on various hardware platforms, and reported overheads in both power draw and hardware area. They also show a promising sign that the hardware overheads of hardware-based detection mechanisms can be kept as low as 1.2% without causing execution slowdowns.

D. Accuracy and Cross Validation

In this section, we focus on the most popular category of mechanisms, where hardware performance counters (HPCs) are used to gather execution data and a machine learning algorithm is used to classify them into safe and dangerous classes. Almost every paper in this category reports accuracies of at least 90%. A notable mention is [32], which claims a 100 percent accuracy for spectre and meltdown.

We did our own study to verify: (1) the reported accuracies can be replicated without extensive tuning and cherry-picking; (2) the reported accuracies apply to a wide range of workloads and attacks; (3) variations in detection frequencies don't cause significant differences in detection accuracy. We used SPEC CPU 2017 as examples of safe workloads and side-channel attack proof-of-concept examples from Google SafeSide to produce ”systems-under-attack” data. Using the same data gathering setup from section IV-A, we classified samples into ”safe and malicious” classes.

We also tested a variety of ML models on the data gathered. We used model presets from the MLJAR-Supervised collection. Each model carries a similar amount of computational complexity, except ”Baseline,” which always guesses the most likely class, and ”Linear,” which is a simple linear logistic regression. ”Ensemble” produces an average of the top three other models. We trained each model for a similar amount of time on CPU and produced out-of-sample classifications. There are papers reporting that ensemble methods produce more reliable detection mechanisms [23] as we were able to confirm in section IV-A.

While the accuracies reported in literature are obtained on out-of-sample data, the training and testing data tend to re-use the same workloads, victims, and attacks. The high accuracy is a good indication that these methods would defend well against attacks they have seen before. However, this does not preclude previously unseen workloads from raising false alarms or previously unseen attacks from passing undetected.

E. Ease of Implementation

Since Cache side-channel attack detection is a relatively newer research field, most papers in the taxonomy go through a very similar approach to their algorithms and counters set. For example [12], [28] both implement very similar monitoring deviations of CPU performance counters to demonstrate how hardware level attacks can be found during run time. The implementation of these is relatively easy, since its mainly statistical analysis, and sometimes implement algorithms, like AES, to monitor the hardware counters.

Some of the more difficult implementations shown are the explainable machine learning ones, which creates an ML framework, such as HPCDR [10], to add on top of the existing machine learning performance counter solutions. The reason for this is that the attacks are detected at a different rate, and can even provide temporal differences of hardware events in timestamps, not just statistics [21].

V. Discussion

We set out to complete this project in an effort to do a survey of numerous side-channel attack strategies from prior research. The aim of current research is to detect the presence of several known malware. In this section, we discuss the types of work towards which we anticipate the field will move in the future.

- Further machine learning Research
- Further Hardware-based Research
- Further Signature-based attack detection Research
- Different Architecture Research

We feel that a number of the topics now being investigated are contributing to the advancement of machine learning in this discipline. The majority of the effort in older studies included in this taxonomy involved manually running and comparing findings [4]. This required a great deal of human input and labor, which would be impractical when checking for several viruses. A notable trend observed was that many contemporary research efforts employ machine learning and machine learning classifiers to identify side-channel threats. Most of them are also considering running many machine learning models to determine which is the most optimal while incurring little performance overhead. This information can then be translated into Deep Learning Models, which can eventually learn to detect any type of virus.

As previously indicated, the recommended Granularity of oversight for the majority of these publications is Operating System-based. We propose that in the future, the focus should be on examining novel implementations, such as at the hardware level, to determine whether any interesting results can be obtained. Given the proximity of the mechanism to the other components, the benefits could include a quicker response time.

One of the most important conclusions from this research is how precise the attacks are. Prior to Meltdown and Spectre being a prominent threat, the majority of research focused on a small sample of malware, such as Trojan Horses. As soon as this transpired, a number of publications initiated research on the Spectre and meltdown viruses. This meant that the vast majority of machine learning models were taught to detect these two specific viruses, making them quite proficient at doing so. As a result, we have discovered that the majority of published papers employ a limited number of malware and attacks to test their study. Signature-based attack detection procedures are essential since it is almost probable that more intrusive and hazardous malware will emerge at some point, and it is vital that our approach can identify even these in the system. Signature-based attack detection, which appears to be one of the future advances in this field for future malware, has received very little attention, as demonstrated in prior sections.

An observable aspect of the current taxonomy is the proportion of work that is x86 Intel/AMD architecture-based. We have observed some development on the ARM and RISC-V architectures, but it is extremely limited. Due to the fact that attack characteristics can vary among architectures, it is crucial that we conduct additional research on all of them. ARM, in particular, should be the focus of the majority of research, as all current Apple Silicon Macs have ARM-based CPUs. Given their market size, they may be susceptible to several side-channel attacks that we have not exhaustively evaluated.

VI. Conclusion

This effort was undertaken to develop a comprehensive taxonomy of the multiple techniques for detecting side-channel attacks on diverse types of hardware. The effect of the Spectre and Meltdown viruses led to the development of several novel mitigating strategies. Experiments on the chip focus mostly on SCA monitoring and assault detection.

Several types of known malware have been discovered through study, and the incorporation of machine learning has proved to be of considerable help to the field’s advancement. In our experimental setting, we reviewed approximately 70 attack detection approaches and the tradeoffs between them. By presenting cost metrics for method comparison and quantifying sensitivity to various system design characteristics, we gave a realistic road map for navigating implementational trade-offs.

VII. Acknowledgments

The work was done through the joint efforts of Xida Ren and Dhruv Pandya. Xida Ren has a vital contribution to this project, and has given a lot of the experimental work needed for this research.

This research is performed with the support of NXP Semiconductors. We thank them for their dedication to hardware reliability and security.

Many of the experimental evaluations are performed on Equinix Metal’s bare metal servers. We thank them for providing idle servers to us free of charge and applaud them for their streamlined user interface and easy provisioning system.

References

[1] B. A. Ahmad, "Real time detection of spectre and meltdown attacks using machine learning."

[2] M. Alam, S. Bhattacharya, and D. Mukhopadhyay, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks," p. 26.

[3] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, "SMoTherSpectre: exploiting speculative execution through port contention," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 785–800.

[4] J. Chen and G. Venkataramani, "CC-hunter: Uncovering covert timing channels on shared processor hardware," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 216–228, ISSN: 2379-3155.

[5] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, "Real-time detection for cache side channel attack using performance counter monitor," vol. 10, no. 3, p. 984.

[6] J. Demme, M. H. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. J. Stolfo, "On the feasibility of online malware detection with performance counters," vol. 41, no. 3, pp. 559–570, MAG ID: 2166844173.

[7] J. Depoix and P. Altmeyer, "Detecting spectre attacks by identifying cache side-channel attacks using machine learning," *Advanced Microkernel Operating Systems*, vol. 75, 2018.

[8] H. Fang, S. S. Dayapule, F. Yao, M. Doroslovacki, and Guru Venkataramani, "Cache-zoomer: On-demand high-resolution cache monitoring for security," vol. 4, no. 3, pp. 180–195, MAG ID: 3033396271.

[9] Hosein Mohammadi Makrani, Zhangying He, Setareh Rafatirad, and Hossein Sayadi, "Accelerated machine learning for on-device hardware-assisted cybersecurity in edge platforms," MAG ID: 4283736170.

[10] A. P. Kuruvila, X. Meng, S. Kundu, G. Pandey, and K. Basu, "Explainable machine learning for intrusion detection via hardware performance counters," vol. 41, no. 11, pp. 4952–4964.

[11] A.-T. Le, T.-T. Hoang, B.-A. Dao, A. Tsukamoto, K. Suzaki, and C.-K. Pham, "A real-time cache side-channel attack detection system on RISC-v out-of-order processor," vol. 9, pp. 164 597–164 612.

[12] C. Li and J.-L. Gaudiot, "Detecting malicious attacks exploiting hardware vulnerabilities using performance counters," in *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, pp. 588–597.

[13] C. Li and J.-L. Gaudiot, "Online detection of spectre attacks using microarchitectural traces from performance counters," in *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, pp. 25–28.

[14] Y. Lyu and P. Mishra, "A survey of side-channel attacks on caches and countermeasures," vol. 2, no. 1, pp. 33–50.

[15] S. Mirbagher-Ajorpaz, G. Pokam, E. Mohammadian-Koruyeh, E. Garza, N. Abu-Ghazaleh, and D. A. Jimenez, "PerSpectron: Detecting invariant footprints of microarchitectural attacks with perceptron," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, pp. 1124–1137.

[16] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "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*. ACM, pp. 1–8.

[17] M. Mushtaq, A. Akram, M. K. Bhatti, R. N. B. Rais, V. Lapotre, and G. Gogniat, "Run-time detection of prime + probe side-channel attack on AES encryption algorithm," in *2018 Global Information Infrastructure and Networking Symposium (GIIS)*. IEEE, pp. 1–5.

[18] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit, "WHISPER: A tool for run-time detection of side-channel attacks," vol. 8, pp. 83 871–83 900.

[19] M. Mushtaq, M. A. Mukhtar, V. Lapotre, M. K. Bhatti, and G. Gogniat, "Winter is here! a decade of cache-based side-channel attacks, detection & mitigation for RSA," vol. 92, p. 101524.

[20] R. Oshana, M. A. Thornton, E. C. Larson, and X. Roumegue, "Real-time edge processing detection of malicious attacks using machine learning and processor core events," in *2021 IEEE International Systems Conference (SysCon)*. IEEE, pp. 1–8.

[21] Z. Pan and P. Mishra, "Automated detection of spectre and meltdown attacks using explainable machine learning," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, pp. 24–34.

[22] Paul C. Kocher, P. C. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, Mike Hamburg, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," pp. 1–19, ARXIV_ID: 1801.01203 MAG ID: 2963311060 S2ID: abef93106038b3be782784ce7ab6109a1798e57d.

[23] G. Perin, . Chmielewski, and S. Picek, "Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis," pp. 337–364.

[24] X. Ren, L. Moody, M. Taram, M. Jordan, D. M. Tullsen, and A. Venkat, "I see dead ops: Leaking secrets via intel/AMD micro-op caches," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, pp. 361–374.

[25] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: a side-channel attack detection tool for tracking prime+probe," in *Proceedings of the International Conference on Computer-Aided Design*. ACM, pp. 1–8.

[26] H. Sayadi, H. Mohammadi Makrani, O. Randive, S. M. P.D., S. Rafatirad, and H. Homayoun, "Customized machine learning-based hardware-assisted malware detection in embedded devices," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, pp. 1685–1688.

[27] M. Taram, X. Ren, A. Venkat, and D. Tullsen, "SecSMT: Securing SMT processors against contention-based covert channels," p. 19.

[28] Z. Tong, Z. Zhu, Z. Wang, L. Wang, Y. Zhang, and Y. Liu, "Cache side-channel attacks detection based on machine learning," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, pp. 919–926.

[29] H. Wang, S. Salehi, H. Sayadi, A. Sasan, T. Mohsenin, P. D. Sai Manoj, S. Rafatirad, and H. Homayoun, "Evaluation of machine learning-based detection against side-channel attacks on autonomous vehicle," in *2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, pp. 1–4.

[30] H. Wang, H. Sayadi, G. Kolhe, A. Sasan, S. Rafatirad, and H. Homayoun, "Phased-guard: Multi-phase machine learning framework for detection and identification of zero-day microarchitectural side-channel attacks," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*. IEEE, pp. 648–655.

[31] H. Wang, H. Sayadi, T. Mohsenin, Liang Zhao, L. Zhao, A. Sasan, S. Rafatirad, and H. Homayoun, "Mitigating cache-based side-channel attacks through randomization: a comprehensive system and architecture level analysis," pp. 1414–1419, MAG ID: 3036243698.

[32] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun, "SCARF: Detecting side-channel attacks at real-time using low-level hardware features," in *2020 IEEE 26th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, pp. 1–6.

[33] H. Wang, H. Sayadi, A. Sasan, S. Rafatirad, and H. Homayoun, "Hybrid-shield: accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks," in *Proceedings of the 39th International Conference on Computer-Aided Design*. ACM, pp. 1–9.

[34] X. Wang, S. M. Chai, Michael Anthony Isnardi, M. A. Isnardi, Sehoon Lim, S. Lim, and R. Karri, "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," vol. 13, no. 1, p. 3, MAG ID: 2319159802.

[35] M. Yan, Y. Shalabi, and J. Torrellas, "ReplayConfusion: Detecting cache-based covert channel attacks using record and replay," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–14.

[36] J. Yu, M. Yan, A. Khyzha, A. Morrison, J. Torrellas, and C. W. Fletcher, "Speculative taint tracking (STT): A comprehensive protection for speculatively accessed data," vol. 40, no. 3, pp. 81–90.

[37] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in *Research in Attacks, Intrusions, and Defenses*, F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, Eds. Springer International Publishing, vol. 9854, pp. 118–140, series Title: Lecture Notes in Computer Science.

[38] L. Zhao, P. Li, R. Hou, M. C. Huang, P. Liu, L. Zhang, and D. Meng, "Exploiting security dependence for conditional speculation against spectre attacks," vol. 70, no. 7, pp. 963–978, conference Name: IEEE Transactions on Computers.