Title
Applied Machine Learning for Analyzing and Defending against Side Channel Threats

Author
Wang, Han

Publication Date
2022

Peer reviewed|Thesis/dissertation

Applied Machine Learning for Analyzing and Defending against Side Channel Threats

By

HAN WANG
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Houman Homayoun, Chair

Seterah Rafatirad

Chen-Nee Chuah

Committee in charge
2022

Abstract

The sharing of hardware components in modern processors helps to achieve high performance and meet the increasing computation demand. Though isolation has been done among users and applications at operating system level, recent research shows that attacks can leverage sophisticated approaches to observe the behaviors of the shared hardware components and infer secrets including password, secret key, etc. Such observations and corresponding attacks are called as side channels and side-channel attacks (SCAs). A number of SCAs have been discovered including Flush+Reload, Flush+Flush, Prime+Probe, Spectre, Meltdown, Fallout, RIDL, ZombieLoad. SCAs have threatened the security of billions of hardware devices, including chips manufactured by Intel, Apple, ARM, etc. Therefore, it is urgent to address the security threats caused by SCAs.

This dissertation pursues the use of machine learning to design effective defense mechanisms and obtain a comprehensive understanding of the side channel threats for emerging applications. In particular, we propose to tackle from three aspects: detection, mitigation and vulnerability analysis.

For detection part, we leverage the microarchitecture level information, i.e. hardware performance counters, to build machine learning-based SCAs detectors. Eventually, we propose two customized machine learning classification models to capture SCAs at real-time and detect zero-day SCAs respectively. As the increase edge devices deployed in the network, we also investigate the machine learning-based detectors against malware and SCAs on autonomous vehicles, mobiles and laptops respectively. We find that hardware performance counters can effectively capture the SCAs with machine learning techniques.

A second aspect of the dissertation is exploring the existing system level and hardware level settings for designing light-weight SCAs mitigation approaches. We find that randomizing the frequency and prefetchers can obfuscate side channel traces and protect against secret leakage. Based on the effectiveness of machine learning-based SCAs detection and randomization-based mitigation, we further developed a detection-mitigation defense approach to further minimize performance overhead incurred by adjusting hardware and

system level parameters.

In the last part of this dissertation, we evaluated the side channel leakage in more general applications which are mostly neglected in the prior side channel research community. We find that hardware performance counters can also be used by attackers to fingerprint websites users visited. Besides, we also discover that the inputs' labels of deep learning models are susceptible to be leaked via side-channel attack, i.e. Flush+Reload. To the best of our knowledge, we are the first group to identify the correlation between label information and side channel observations, highlighting the importance of reexamining the side channel vulnerability in general applications.

Acknowledgments

After completing my bachelor's degree, I went for a job directly and gradually realized I wanted to explore topics in-depth. Luckily, I got accepted into the graduate school and worked with my Ph.D. advisor, Dr. Houman Homayoun. When I look back on the decision, I think this might be one of the most intelligent decisions I have ever made. In the past three years at Davis, I had the opportunities to work on research topics I am passionate about and received numerous support from mentors, collaborators, and excellent peers. Without them, I will definitely not be able to complete the research work and gain the fruitful research training I had. I want to thank them individually for supporting and helping me.

First of all, I would like to thank my Ph.D. advisor, Dr. Houman Homayoun. Since I had no prior research experience when I started my Ph.D., he offered me enormous guidance to help me adapt to the change from industry to academia. Besides, his passion and vision for research always motivated me to focus on research problems and made the journal enjoyable. I am also grateful that Dr.Houman Homayoun supports my professional choice and provides me with extensive help. I would achieve none of my work without his support and mentorship.

I am thankful to other faculty members I collaborated during my Ph.D., including but not limited to Dr. Chen-Nee Chuah, Dr. Zubair Shafiq, Dr. Avesata Sasan, Dr. Khaled Khasawneh, and Dr. Sai Manoj P D. They provided valuable feedback, feedback, and inspiration over the years that I studied at University of California Davis and George Mason University. I am also thankful to the dissertation committee members: Dr. Setereh Rafatirad and Dr. Chen-Nee Chuah, for providing helpful comments and invaluable feedback about this dissertation.

I also want my peers from ASEEC Lab, including Hossein Sayadi, Hosein Mohammadi Makrani, Chongzhou Fang, Gaurav Kolhe, Ning Miao, Tyler Sheaves, Ryan Tsang, Asmita Asmita, Katayoun Neshatpour, and Maria Malik. They definitely enriched my Ph.D. journey, and I am fortunate to work with them.

Lastly, I would like to express my gratitude for the support and encouragement from my parents and older sister. I could imagine how hard it is for them not to be able to see me in person for years. Their tremendous understanding is always one of my strongest motivations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern computers leverage the sharing of hardware components to achieve high performance and meet the increasing computation demand. The isolation among users and applications is achieved at the operating system level to ensure computer security. Though effective, they also bring new attacking surfaces, i.e., side channels, for a malicious program to observe the benign applications' hardware-level behaviors and infer secrets. In the past decade, researchers have found a number of attacks exploit the attack surfaces and collect side channels, including Flush+Reload [2], Flush+Flush [3], Prime+Probe [1], Spectre [4], Meltdown [5], Fallout [6], RIDL [7], ZombieLoad [8]. Such attacks, termed side-channel attacks (SCAs), have threatened the security of billions of hardware devices, including the ones manufactured by Intel, AMD, and Apple. Compared to malware or software-based attacks, such attacks can pose a significant security threat on a broader range of systems and are more difficult to eliminate due to the invisible behavior and passive nature of SCAs [9]. Even worse, researchers have found that the target of SCAs is not limited to encryption applications [1–3] but also more general applications, like deep learning-enabled applications [10–13]. For example, Hong et al. [11, 12] exploit the cache-based SCAs, Flush+Reload, to steal information during the inference phase to reconstruct the crucial architectures of DNNs. It proposes an algorithm that generates candidate computational graphs from Flush+Reload observations, and the parameter estimation process removes incompatible candidates with 0% error for MalConv [14] and ProxylessNAS [15]. Thus, there is an emerging need to address the security threats posed by such attacks in modern computer systems.

Hence, this dissertation aims to leverage machine learning (ML) techniques to explore

effective defense mechanisms and understand the side-channel threats comprehensively. In this dissertation, we have three main parts: 1) investigate the effectiveness of using hardware-level features and ML techniques for detecting SCAs; and 2) explore the light-weight mitigation strategies with minimal hardware redesign and performance overhead; and 3) analyze the side-channel observation and their potential leakage in general applications. In the first part, we presented that leveraging ML techniques and hardware-level features can build adequate SCAs detectors for real-time SCAs capturing and zero-day SCAs detection. In the second part, we explored the opportunity of randomizing system-level and hardware-level settings to obfuscate side-channel traces and protect against secrets leakage. Based on the effectiveness of ML-based SCAs detection and randomization-based mitigation, we developed a detection-mitigation defense approach to minimize further performance overhead incurred by adjusting hardware and system level parameters. In the last part of this dissertation, we evaluated the side-channel leakage in more general applications, which are mostly neglected in the prior side-channel research community. We find that attackers can also use hardware performance counters to fingerprint websites users visit. Besides, we also discover that the inputs' labels of deep learning models are susceptible to being leaked via side-channel attack, i.e., Flush+Reload.

The remainder of this chapter is organized as follows. Section 1.1 describes the research problems this dissertation tries to solve. Then we will summarize the contribution of this dissertation in Section 1.2. Finally, we will present the organization of this dissertation in Section 1.3.

1.1 Motivation

This dissertation pursues solutions for defending against side-channel attacks from three aspects: 1) investigate the effectiveness of using hardware-level features and ML techniques for detecting SCAs; and 2) explore the light-weight mitigation strategies with minimal hardware redesign and performance overhead; and 3) analyze the side-channel observation and their potential leakage in general applications. In the first aspect of the dissertation, we present the effectiveness of ML-based SCAs detectors. In particular, real-time attacks detectors have a high chance of experiencing a high false positive rate, significantly decreasing the credibility of attack detection predictions. At the same time, the new variants of side-channel attacks challenge the effectiveness of existing detectors. This dissertation explores designing customized ML algorithms for building practical SCAs detectors

to countermeasure the two challenges.

Since SCAs exploit hardware vulnerabilities, most mitigation approaches demand costly hardware design. Hence, the second aspect of the dissertation investigates the possibility of adjusting hardware components settings to change side-channel traces. Then, attackers cannot recover secrets due to the polluted side-channel traces.

The last part of the dissertation is to analyze the side-channel threats in general applications. Most prior research on side-channel threats targets encryption applications due to public computation algorithms and the correlation between computation and secrets. However, the general applications are also susceptible to side channel threats, while the correlation between computation and secrets is more challenging to identify. The wide deployment of such general applications threatens computer system security and users' privacy. In this dissertation, we analyze the side-channel observations on two types of general application, i.e., web browsers and deep learning-based applications. We employ ML to identify the correlations between side-channel traces and secret information.

The following section will present the research problems that this dissertation plans to address and the limitation of state-of-the-art works.

Machine Learning-based Attacks Detection

Prior studies on detecting microarchitectural side-channel attacks leverage the ML algorithms on the low-level microarchitectural data captured by Hardware Performance Counter (HPCs) registers to detect the known SCAs [9, 16–19]. The ML-based SCAs detectors have demonstrated promising results in determining the side-channel attacks with lower latency ranging from several milliseconds to seconds. For instance, the work in [16] proposes an HPC-based monitoring model to detect the SCAs by using the performance counters events collected by running both victim and SCA applications. The mechanism deploys the correlation metric between the events of victims' and attacks' HPC traces. Another work proposed in [19] profiles the victim's application under two different scenarios, i.e., applications with no attacks, and applications with attacks. Although this detection technique helps mitigate various SCAs, it can not detect unknown SCAs.

There has been much progress in terms of and securing the processors against various microarchitectural SCAs in the last few years. However, there are still two and major challenges involved with contemporary SCA detectors. First, the fast-paced development of the emerging

intricate SCAs to circumvent the current detection techniques has not been properly addressed. Hence, the existing techniques are not able to detect and identify the unknown (zero-day) attacks. Second, the existing side-channel attacks detectors limit their study to detecting whether the running application is "under attack" condition or not. In particular, such detectors provide no information on which type of vulnerability the attack applications exploit. Knowing the type of SCAs ahead of time could facilitate crafting an effective mitigation technique to alleviate the influence of side-channel attacks on the performance of the target system.

To address the limitations of the existing SCA detectors, this work we propose *Phased-Guard*, a multi-phase ML framework to accurately detect and identify both known and unknown attacks at run-time using the most prominent microarchitectural features.

Side-Channel Vulnerability Defense

Timing-based cache SCAs [1, 2, 20] can be launched by the attacker remotely (e.g., attacks can even occur in cloud environments). Such attacks exploit the accessing time gap between the on-chip caches and main memory, and collect cache hit/miss traces based on various accessing times. Hence, the attacks can infer sensitive information according to cache traces and the knowledge captured from the cryptographic algorithm. There exists a number of cache-based SCAs proposed in prior studies [1–3, 21] causing a substantial threat to the security of modern computer systems. Due to the invisibility, feasibility, and capability to expose and extract the secret keys in the cache-based SCAs, there is an urgent need to address the security risks posed by such attacks in present computer systems as well as legacy systems [2, 3].

Prior works on cache-based side-channel attacks mitigation can be categorized into two main designing principles including cache partitioning [22], and randomization-based techniques [23]. Cache partitioning methods have been proposed to isolate cache usage between different programs to prevent the attackers from observing the victims' data access patterns and stealing the confidential information stored in the cache memory [22–24]. In general, the cache partitioning techniques divide the cache memory into different zones for different application processes statically or dynamically. As a result, attack applications do not have access to observing cache access of users' applications. Another approach that was proposed to mitigate the impact of SCAs is based on randomization techniques [25, 26] in which they attempt to randomize the memory-to-cache mappings. These

methods were mainly proposed in the architecture community as a solution to protect future architectures. They are not applicable to current and legacy architectures since they require hardware redesign efforts.

In response, this work proposes a comprehensive system and architecture level randomization methodology to efficiently mitigate the impact of side-channel attacks on last-level caches eliminating the need to modify the cache memory architecture.

Side-Channel Vulnerability Evaluation with Machine Learning

Most existing research takes applications of cryptography as victims of SCAs [1–3] while general use applications have been proven to be subject to side-channel leakage [10]. For example, a deep learning-enabled application contains sensitive information, including architectures, trained weights, and labels of input data which can be the stealing target of SCAs. Compared to applications of cryptography, changing the setting of general use applications is more complicated. Taking deep neural networks as an example, the architectures or trained weights cost a fortune to build, while the encryption application RSA can change secret keys after discovering being attacked.

Secondly, applications like deep learning have been increasingly used for critical domains, from financial decisions [27], energy control [28], autonomous systems [29], medical treatment [30], etc. Their label information either contains critical information or impacts significant decisions, which attackers can steal and make an undesired profit out of them or conduct crimes based on them. In the finance domain, attackers can misuse the investment suggestions stolen from victim DNNs. Energy controlling systems [28] leverage DNNs to design the optimal online power control policy while the attacker with label information can take advantage of the policy information to deliberately overload the energy network and cause a denial-of-service attack on customers.

1.2 Overview of the Dissertation

The research of this dissertation focuses on addressing the security threats brought by side-channel attacks with ML techniques. Chapter 2 exhibits the related works in the research problems this dissertation addresses, including ML-based SCAs detection, light-weight mitigation, and side-channel vulnerability analysis. The rest of the dissertation will be organized into three parts as detailed below.

Part 1: Machine Learning-based SCAs Detection

This part tries to address the challenges by proposing effective ML-based detectors with the use of hardware-level features. In response, this part includes two projects to solve the real-time SCAs capturing and zero-day SCAs detection respectively by collecting run-time behaviors of victims applications and designing customized ML classification models.

Detecting side-channel attacks at real-time using low-level hardware features [31]: Side-Channel Attacks (SCAs) are powerful attacks compromising the security of modern computer systems have considered collecting hardware events of both victim applications (cryptographic application, e.g. RSA, AES and etc.) and attack applications. However, in such techniques, the attack HPCs data can be easily manipulated and/or corrupted, resulting in misleading the SCA detection mechanism. Furthermore, the prior works have explored the suitability of a limited number of ML algorithms in detecting SCAs without examining the instance level false alarm rate. As we show in this work is a more critical evaluation metric for real-time detection techniques. In response, in this paper, we propose *SCARF*, a ML-based real-time side-channel attack detection methodology using low-level hardware features. To this aim, we first only monitor the victim applications' behavior using the HPC features and analyze the captured low-level traces of the victim applications under no attack and attack conditions to avoid manipulating attackers' HPCs. Next, a wide range of ML classifiers with customized HPC features are implemented to determine the most effective ML technique for detecting SCAs in real-time while improving accuracy and reducing instance-level false alarm rate of ML-based SCA detectors. Lastly, the False Alarm Minimization (FAM) technique is proposed to reduce the instance level false positive rate of the ML-based SCA detectors. The experimental results indicate that the *SCARF* methodology can obtain up to 100% attack detection accuracy with 0% instance level false alarm rate for detecting SCAs.

Hybrid Dynamic Time Warping and Gaussian Distribution Model for Detecting Emerging Zero-Day Microarchitectural Side-Channel Attacks [32]: Microarchitectural Side-channel Attacks (SCAs) benefit from emerging hardware vulnerabilities in modern microprocessors to steal critical information from users, posing great security threats to computer systems. Several recent studies have focused on using low-level features captured from built-in Hardware Performance Counter (HPC) registers to implement accurate ML-based SCAs detectors. Nonetheless, existing

ML-based SCAs detectors required prior knowledge of attacks to detect the pattern of side-channel attacks using various microarchitectural features. In particular, the existing solutions have ignored to address the challenge of detecting sophisticated unknown (zero-day) SCAs at run-time which is a more challenging issue in today's computer systems. In addition, prior works analyzed a limited number of ML classifiers without thoroughly evaluating the detectors' detection effectiveness and computational complexity. In response, we propose *HybriDG*, a hybrid light-weight model consisting of Dynamic Time Warping (DTW) followed by a Gaussian distribution model to detect both known and unknown emerging SCAs at run-time accurately. Our experimental results demonstrate that *HybriDG* achieves 100% detection accuracy for known attacks and 99.5% detection accuracy for unknown attacks, which is significantly outperforming traditional ML algorithms, deep learning, and time series classification models by up to 80% for unknown and 8% known attack detection.

Evaluation of Machine Learning-based Detection against Side-Channel Attacks on Autonomous Vehicle [33]: Autonomous vehicles are becoming increasingly popular, but their reliance on computer systems to sense and operate in the physical world has introduced new security risks. Recent studies have shown that using Cache-based Side-Channel Attacks (SCAs) could infer sensitive users' information (e.g., which route the user is taking) highlighting significant vulnerability posed to today's computer systems. As a result, it is crucial to propose effective detection mechanisms against emerging microarchitectural SCAs on autonomous driving systems. In response, this work identifies the threat model and victim applications of autonomous driving systems. Next, we explore the suitability of various ML-based classifiers trained by information collected from built-in hardware performance counter registers available in modern autonomous vehicle systems. To this end, various supervised ML models are implemented for cache-based SCAs detection and precisely compared and characterized in terms of detection accuracy, robustness, and latency of the detection. Our experiments conducted on an Intel Xeon, which Waymo autonomous driving vendor uses, demonstrate that J48 achieves 99.5% accuracy with the highest efficiency compared with other investigated models.

Enabling Micro AI for Securing Edge Devices at Hardware Level [34]: Compared to servers and workstations, edge devices are more likely to experience the limitation of computation and energy to defend against attacks. To address the challenge, this work proposes an accurate and cost-efficient micro AI enabled countermeasure for securing modern edge devices against emerging

cyber-attacks, i.e., malware and Side-Channel Attacks (SCAs) at the hardware level by monitoring applications' Hardware Performance Counter (HPC) features. To realize a run-time ML-based solution that relies on limited available HPCs in modern edge processors, we first identify the most prominent HPC events for accurate attack detection with the aid of an effective feature selection method. Next, various standard ML classifiers are implemented for effective and accurate run-time hardware-assisted malware and side-channel attacks detection. They are compared and characterized in terms of detection accuracy, F-measure, robustness, latency, power consumption, and hardware overheads. Experimental results demonstrate that the J48 classifier achieves the highest detection rate (F-measure) for both malware and SCAs detection with 0.917 and 0.987, respectively, with relatively negligible latency and area overhead as compared to complex models making it a suitable algorithm for enabling an efficient hardware-assisted micro AI countermeasure in edge devices.

Part 2: Light-weight SCAs Mitigation

The second part of this thesis tries to eliminate the side-channel vulnerability by polluting attackers' observation with hardware redesign. To achieve this, we explore the effectiveness of adjusting the existing system-level and hardware-level settings, i.e., frequency and prefetchers. And then, we propose a detection-mitigation framework to minimize the performance overhead further while securing victim applications from side-channel leakage. The overview of the two projects is introduced below.

Mitigating Cache-Based Side-Channel Attacks through Randomization: A Comprehensive System and Architecture Level Analysis [35]: This work proposes a light-weight system and architecture level randomization technique to effectively mitigate the impact of side-channel attacks on last-level caches with no hardware redesign overhead for current and legacy architectures. To this aim, by carefully adapting the processor frequency and prefetchers operation and adding the proper level of noise to the attackers' cache observations, we attempt to protect the critical information from being leaked. The experimental results indicate that the concurrent randomization of frequency and prefetchers can significantly prevent cache-based side-channel attacks with no need for a new cache design. In addition, the proposed randomization and adaptation methodology outperforms the state-of-the-art solutions in terms of the performance and execution time by reducing the performance overhead from 32.66% to nearly 20%.

Accurate and Efficient Cross-Layer Countermeasure for Run-Time Detection and Mitigation of Cache-Based Side-Channel Attacks [36]: We propose *Hybrid-Shield* in Chapter 4, an accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. For the detection stage, microarchitectural information of victim under attack and under no attack conditions are collected for training ML classifiers. For the mitigation stage, *Hybrid-Shield* adapts hardware prefetchers and scales processor frequency to increase the noise level in observed cache access pattern attacks to induce secret information. The experimental results indicate that *Hybrid-Shield* can achieve 100% detection rate with 0% false alarm rate and detected attacks' error rate increases from less than 5% to above 35% with only 15% performance overhead.

Part 3: Machine Learning-enabled Side-Channel Vulnerability Analysis

Besides encryption applications, more general applications also have a potential risk of being exposed to side-channel attacks and suffering from sensitive information loss. This part of the dissertation aims to analyze the side-channel vulnerability of general applications, i.e., web browsers and deep learning models.

Accurate and Efficient Machine Learning-Based Website Fingerprinting Attack through Hardware Performance Counters [37]: Users' website browsing history contains sensitive information, like health conditions, political interests, financial situations, etc. In order to cope with the potential website behavior leakage and enhance the browsing security, some defense mechanisms such as SSH tunnels and anonymity networks (e.g., Tor) have been proposed. Nevertheless, some recent studies have demonstrated the possibility of inferring website fingerprints based on important usage information such as traffic, cache usage, memory usage, CPU activity, power consumption, and hardware performance counters information. However, existing website fingerprinting attacks demand a high sampling rate which causes high performance overheads and significant network traffic, and/or they require launching an additional malicious website by the user, which is not guaranteed. As a result, such drawbacks make the existing attacks more noticeable to users and corresponding fingerprinting detection mechanisms. In response, in this work, we propose *Leaked-Web*, a novel accurate and efficient ML-based website fingerprinting attack through processor's Hardware Performance Counters (HPCs). *Leaked-Web* efficiently collects hardware

performance counters in users' computer system at a significantly low granularity monitoring rate and sends the samples to the remote attack's server for further classification. *Leaked-Web* examines the web browsers' microarchitectural features using various advanced ML algorithms ranging from classical, boosting, deep learning, and time-series models. Our experimental results indicate that *Leaked-Web* based on a LogitBoost ML classifier using only the top 4 HPC features achieves 91% classification accuracy outperforming the state-of-the-art attacks by nearly 5%. Furthermore, our proposed attack obtains a negligible performance overhead (only <1%) which is around 12% lower than the existing hardware-assisted website fingerprinting attacks.

Stealthy Inference Attack on DNN via Cache-based Side-Channel Attacks [38]: Most prior works focus on side-channel vulnerabilities in cryptography applications, including RSA, AES, and others. However, emerging applications containing sensitive information are also susceptible to side-channel leakage. This work takes the deep neural networks as victim applications and employs ML techniques to build the correlation between observation from the cache-based SCA Flush+Reload and sensitive information. We consider two attacking scenarios: binary attacking identifies specific sensitive labels and others, while multi-class attacking targets recognize all classes of victim DNNs provide. Among all six deep neural network models, including DenseNet 121, DenseNet 169, VGG 16, VGG 19, MobileNet v1, and MobileNet v2, experimental results show that MobileNet v1 is the most vulnerable one and experiences 99% and 75.6% attacking success rates for binary and multi-class attacking scenarios, respectively. It highlights the necessity of rethinking the side-channel vulnerabilities in emerging deep learning-based applications.

Lastly, Chapter 7 discusses the potential future works.

1.3 Organization

This dissertation focuses on side channel analysis and defense with machine learning techniques and is organized as below.

Chapter 2 surveys existing works for SCAs detection, mitigation, and side-channel vulnerability evaluation.

Chapter 3 introduces two customized machine learning-based SCAs detectors, *SCARF* and *HybriDG* respectively for addressing the real-time SCAs capturing and zero-day SCAs detection.

Chapter 4 further investigates the feasibility of leveraging hardware-level behaviors and

machine learning techniques to build an effective attacks detector on autonomous vehicles, mobile, and laptops.

Chapter 5 first demonstrates the effectiveness of randomizing system-level and hardware-level settings for obfuscating victim applications' behaviors and securing sensitive information. Based on the finding, we propose *Hybrid-Shield* which contains a detection module and a mitigation module to reduce the performance overhead incurred by adjusting the settings of frequency and prefetchers.

Chapter 6 presents the side-channel vulnerabilities that existed not only in applications of cryptography but also in general applications and emerging deep learning-based applications. It utilizes machine learning classification algorithms to examine the correlation between sensitive information and side-channel observations.

Chapter 7 discusses future directions and conclude this dissertation.

Chapter 2

Background

This chapter introduces the related works of this dissertation. Section 2.1 will introduce the related works in ML-based SCAs detection with the use hardware-level features, i.e. hardware performance counters (HPCs). Then we review the previous mitigation works in Section 2.2. Lastly, we will review the side-channel vulnerability analysis on emerging applications, including website fingerprinting and deep learning leakage.

2.1 Hardware-assisted Side-Channel Attacks Detection

Chiappetta et al. [16] uses HPCs and then applies and compares three different detecting attacks: finding a correlation between victims and attacks; building supervised ML models based on HPCs from victims and attacks; detecting anomalies by validating attack HPCs as samples and other processes as outliers. This work can achieve real-time cache-based SCAs. This work takes Flush+Reload as an example and evaluates the detection effectiveness in terms of F Score and confidence.

Unlike the prior work, CloudRadar [9] proposes a detection system for a cloud-based environment targeting Flush+Reload, Prime+Probe. It first monitors the HPCs of cryptography applications and trains a dynamic time warping ML model which identifies if an application is a cryptography application under the victim virtual machine. If it is, the HPCs from the untrusted VM would be collected and correlated with the protected VM's sensitive operations. According to experimental results, the proposed detection system shows a high true positive (100%) and close to 0 false positive rate when the window size increases to 5, meaning 5.1 milliseconds.

In addition, the work in [39] proposes a multi-layer detection approach targeting identifying the existence of the abnormal process (the attack) and outputting the possible type (like cache-based or branch prediction based) of side-channel-attack for the abnormal process. To this aim, it profiles processes with low-level hardware events using Linux-based perf event API. It then analyzes the HPCs with One-Class Support Vector Machine (OC-SVM) to identify if the process is normal or abnormal. Once it is recognized as abnormal, the HPCs would be sent to the pre-trained classifiers (Adaboost, Random Forest, Naive Bayes, and SVM) and gives the category of the process events, like cache-based, branch-based. The detection system correlates the anomaly process and the encryption applications with the Fast Dynamic Time Warping (fast-DTW) algorithm. If the similarity is substantial enough, the process will finally be predicted as a side-channel attack with a specific type.

The work in [40] adopts an unsupervised learning anomaly detection (Gaussian anomaly detection) to detect potential malicious VMs with side-channel attacks in IaaS (infrastructure as a service) platform. The HPCs monitoring part leverages Intel Cache Monitoring Technology (CMT) to profile the microarchitectural behaviors. It employs Prime+Probe as a case study, and the effectiveness is evaluated in terms of accuracy, F score, and roc curve. The experimental results show that the proposed detection works very well when the system does not have a compute-intensive workload (CIW) while it gives false positives when CIW is present.

To cope with the newly side-channel attack of Spectre [20], the work in [41] trains a neural network with the hardware performance counters to identify a Spectre attack. The whole dataset is split as 90%-10% for training and testing respectively, and the experiment shows over 99% accuracy and 0.972 F score. The high detection accuracy indicates that the cache patterns of Spectre are very distinctive. Compared to prior works, this work also evaluates the performance overhead caused by HPCs monitoring and finds that the used sampling interval of 100 milliseconds in this work only causes 4-5% performance overhead of the machine it was running on. One drawback of this work is that it only uses one Spectre variant while different side-channel attacks have different cache patterns. Since the trained detection model has not met such a pattern before, the proposed detection might not be able to detect.

Compared to prior works profiling both victim and attack applications, [17, 42] monitors the hardware performance counters of victim applications only (like RSA and AES) under SCAs and normal scenarios. Two state-of-the-art SCAs (Flush+Reload and Flush+Flush) are included

in this work. PAPI-based HPCs monitoring tool is leveraged, and roughly 1 million samples are collected where a K-fold cross-validation is conducted to evaluate the effectiveness of the proposed detection system. This work investigates the accuracy of no load, average load, and full load to simulate the realistic workload of computer systems with three different types of classifiers: Linear Discriminant Analysis (LDA), Logistic Regression (LR), and Support Vector Machine (SVM). The experiment shows that the detection accuracy for Flush+Reload is 99.51%, 99.50%, and 99.44% under no, average, and full load conditions within 1% completion of an RSA encryption operation. For Flush+Flush, the detection accuracy is 99.97%, 98.74%, and 95.20%detection accuracy for no load, average load, and full load conditions, respectively, at the cost of 12.5% completion of an attack on AES encryption round.

Similarly, CacheSheild [19] also profiles victims under attacks, and under normal conditions to identify the existence of SCAs. This work adopts a change point detection algorithm (CPD) which is mostly used to construct the commonly known quick detection models. Three classical side-channel attacks: Flush+Reload, Flush+Flush, and Prime+Probe are the detection target of this work. L3_Misses is selected based on the understanding of SCAs, and evaluation is conducted on both KVM and VMware platforms. Yahoo Cloud Serving Benchmark, Video Streaming, and Randmem Benchmark are included to investigate the influence of other running workloads. The detection speed ranges from 4 ms to 13 ms, while the false positive rate ranges from 0.1% (AES with VMware) to 24.4% (RSA with KVM running together with Randmem Benchmark).

Though effective, the above works cannot address the high false positive rate in real-time detection and zero-day SCAs. This dissertation aims to use customized ML models to address the two challenges.

2.2 Side Channel Mitigation

This section presents the latest studies on side-channel attacks mitigation and securing the computer systems against information leakage caused by side-channel attacks. As mentioned earlier, in order to tackle the challenge of information leakage due to side-channel attacks, software-based techniques are proposed to mitigate side-channels caused by shared hardware resources. STEALTHMEM [43] presents a system level protection mechanism against cache-based SCAs in the cloud. The proposed technique manages a set of locked cache lines per core and avoids victim

data being evicted from the cache by the attack. Intel's Software Guard eXtensions (SGX) [44–47] is developed to isolate memory between different applications in which each application will be executed in a secure container named enclave where other processes including kernel, hypervisor, and other privileged code can not obtain access to the data in the enclave. The majority of such protection mechanisms attempt to reduce or remove sharing resources, mainly cache, and memory, to defend security-sensitive applications from attacks at the cost of higher resource requirements and performance degradation. However, recent research [48–51] has shown that applications running within enclaves are still under SCA threat highlighting the necessity of proposing efficient approaches to mitigate such attacks. The work in [52] proposes to conduct transformation in a compiler back-end, which removes key-dependent control flow from x86 programs. However, it lacks solid evidence to show the generality of the methodology and its application across different general-purpose applications.

Other works propose designing new architectures to eliminate the current security threat posed by side-channel attacks. Several recent works have proposed to modify cache hierarchy or cache memory architecture to mitigate SCAs. Cache partitioning techniques [22,24,53] are proposed to mitigate cache-based side-channel attacks by statically or dynamically partitioning cache memory for each application process, thereby SCAs are not able to observe "side-channel information" of victim applications. Another approach employs access randomization [23, 25, 26] which primarily randomizes cache interference, re-maps cache indices, or replaces demand fetch with random cache fill to eliminate security vulnerabilities in the hardware architecture. However, such works require new designs of cache and cache memory translation architectures which pose extra design costs and can not be applied to legacy systems.

A recent work [54] on SCA mitigation has proposed to scale frequency to hide the victim's cache trace. In particular, it presents a general and elastic protection scheme against SCAs in the cloud environment. It requires purchasing a higher clock rate for protected VM, and it lowers the frequency of suspected malicious VM when security-critical operations happen. While this method effectively addresses Flush+Reload mitigation, it introduces a considerable performance overhead of more than 33.7% to the system. In addition, the authors limit their study only to frequency randomization to pollute cache trace. Also, their method requires the knowledge of the processing core where victim applications and malicious applications reside, which is difficult to obtain in a

local environment.

Unlike the discussed prior works on cache-based side-channel attack mitigation, our dissertation presents an effective solution for today as well as future computer systems by proposing a comprehensive system and architecture level randomization methodology to efficiently mitigate the impact of side-channel attacks on last-level caches eliminating the need to modify the cache architecture with no requirement for hardware redesign effort.

2.3 Side-Channel Vulnerability Assessment

2.3.1 Website Fingerprinting

Protecting browsing history has emerged recently as a crucial concept to ensure the preservation of users' privacy. In response, [55, 56] are proposed to leverage SSH based protection methods; [55] encrypts and authenticates messages in one session, and [56] adds cover traffic conservatively while maintaining high levels of security. Similarly, Tor project [57] is one of the most popular traffic transmission approaches, where messages are not directly routed to the receiver but encrypted and forwarded according to ephemeral paths of an overlay network. Though such works have made notable progress, there are still a number of attacks that could bypass such protection mechanisms and extract users' browsing history.

Some recent website fingerprint attacks exploit the clients' machine information when visiting different websites, like memory footprint [58], storage [59], etc. To obtain the information, some attacks prepare a malicious website for users to visit, or a local malware to be launched on the target host. For example, [60] launches a Prime+Probe attack to measure cache occupancy through a malicious website. Then, a deep learning method is leveraged to classify websites and recover users browsing history. Other works such as [58] samples the memory footprint of browsers through the procfs file system in Linux. To defend against such attacks, [61] proposes ML-based syntactic-semantic approach that detects browser fingerprinting attacks' behaviors by incorporating both static and dynamic JavaScript analysis. [62] proposes to monitor the running Web objects on the user's browser and collect fingerprinting related data. Then, it analyzes them and searches for patterns of fingerprinting attempts. Though effective, they only work for attacks deployed through malicious websites. Furthermore, advanced attacks [63] can be deployed in native code

and bypass such detection systems. [64] randomizes properties, like offsetHeight and plugins, to the JavaScript environment, which generates different fingerprints even for the same website and increases non-determinism for attackers. However, randomization is complex and can change the visual appearance of websites.

2.3.2 Deep Learning Leakage

Privacy issues in DNNs have raised increasing attention from both industry and academia. Hong et al. [11, 12] exploit the cache-based SCAs, Flush+Reload, to steal information during the inference phase to reconstruct the crucial architectures of DNNs. It proposes an algorithm that generates candidate computational graphs from Flush+Reload observations, and the parameter estimation process removes incompatible candidates with 0% error for MalConv and ProxylessNAS-CPU. Yan et al. [65] take advantage of the DNNs' reliance on Generalized Matrix Multiply (GEMM) and employs Prime+Probe and Flush+Reload to obtain DNNs' architectures. It reduces the search space from 5.4×1012 to 16 for VGG and 6×1046 to 512 for ResNet-50, respectively. Hua et al. [66] investigate the leakage from memory and other side channels on hardware accelerators even with secure techniques. They find that the memory access patterns can enable reverse-engineering of the structures and weights of CNN models.

Xiang et al. [13] illustrate an attack that extracts power traces on FPGA-based accelerators to reconstruct the input image. It filters out noises and distortion in power measurement with low-pass filters, power, and curve fitting. The capability of the attack is evaluated in the hand-written digits of the MNIST dataset [67], achieving 89% accuracy. Dong et al. [68] measure the execution time of floating-point multiplications from the power consumption traces, which are used to infer the pixel values of images without the knowledge of neural network parameters. It shows 96.2% accuracy for the MNIST dataset. Luo et al. [10] show that using the cache-based SCA, Prime+Probe, to extract the cache access patterns can help to reveal the route or the location of a vehicle with the adaptive Monte-Carlo localization (AMCL) algorithm. It builds the correlation between the cache access pattern observed by Prime+Probe and the label information with statistical learning models.

Chapter 3

Machine Learning-based Side-Channel Attacks Detection via Hardware Performance Counters

The past decades have witnessed a significant boost in the complexity of computing systems to support the increasing computation and performance demand. To this aim, various components (e.g., cache memories, branch predictors, out-of-order execution units) are implemented in modern processors to minimize the CPU stalls and enhance the performance. Despite the provided performance benefits, these solutions could cause new microarchitectural vulnerabilities that have been exploited by new types of attacks that observe side-channel information by causing interference. Microarchitectural Side-Channel Attacks (SCAs) primarily exploit hardware vulnerabilities to infer sensitive and confidential information [2, 3]. The proliferation of computing devices in mobile computing and Internet-of-Things (IoT) domains further exacerbates the impact of emerging cybersecurity threats implying the necessity of protecting legitimate users from these attacks. Hence, an emerging need exists to address the security risks and challenges posed by such harmful attacks, calling for effective and low-cost security countermeasures to accurately identify SCAs with minor overhead. In response, this chapter employs ML techniques and hardware-level features to design a real-time detector and a zero-day attacks detector to secure systems from the threats of SCAs.

3.1 Detecting side-channel attacks at real-time using low-level hardware features

Side-channel attacks (SCAs) primarily target inferring sensitive and confidential information from a computer system through measurement and analysis of physical parameters [23, 69, 70]. Cache-based SCAs are one of the most common side-channel attacks that can be launched by the attacker remotely and require no physical access [2, 20]. As a result, there exists an emerging need to address the security risks and challenges posed by such harmful attacks, demanding an effective SCAs detection methodology that can accurately identify SCAs threats with minor overhead.

Prior works on side-channel attack detection such as [9, 16, 41] propose the use of microarchitectural pattern analysis captured through Hardware Performance Counters (HPCs) to detect the SCAs with latency by order of ranging magnitude from milliseconds to seconds. For instance, the work in [16] proposes to detect the SCAs with the usage of both victim and attack applications' HPCs traces. Then based on the obtained HPCs, the correlation between the HPC events of victims' and attacks' traces will be evaluated. Similarly, in [9] the authors present CloudRadar, which aims at detecting cross-VM side-channel attacks by making use of HPC patterns. Undoubtedly, the prior detection works have made some progress in detecting the SCAs. However, they fall short in addressing the challenges defined above and several drawbacks, as demonstrated below.

- *Lack of Robustness:* Our comprehensive analysis shows that the majority of previous works on side-channel attacks detection jointly correlate the HPCs traces of victim and attack applications [9, 16]. However, recent studies [71, 72] have demonstrated that current HPC features monitoring methods suffer from the overcounting issue that creates the opportunity for attackers to manipulate HPCs data by slightly changing SCA applications. Hence, current detection techniques relying on the HPCs data of attack applications are facing notable security threats.

- *Limited Machine Learning Classifiers:* A wide range of classification and anomaly detection techniques are developed by applying ML techniques. However, existing works in particular on SCA detection have primarily focused on one or a few ML techniques for the purpose of attack detection and classification [9, 16]. Such an analysis leaves a void in terms of the performance of attack detection, as various ML classifiers yield different performance in detecting various types of attacks [73].

- *High False Alarm Rate:* Prior studies on real-time HPC-based SCAs detection have

neglected to examine the instance level (a complete temporal sequence of victim applications' HPCs) false positives of HPC data and have only evaluated the SCAs detectors based on the interval level (a sub-sequence of victim applications HPCs) false positive rate [9, 16]. However, SCA detection based on the capturing intervals of HPCs data is biased to "under attack" conditions. The Victim under No Attack (VNA) requires all the captured HPCs intervals to be classified correctly by the ML-based SCA detector to achieve a correct prediction while Victim under Attack (VA) requires only one interval classified correctly to achieve a correct prediction.

To address the challenges above, in this work, we propose *SCARF*, a unified and accurate ML-based real-time side-channel attack detection methodology using low-level hardware features. To this end, we first demonstrate the validation of detecting side-channel attacks based on the victim application HPCs data to avoid manipulation of the attacker's HPC by executing the victim applications on an isolated processing core. Under such isolated circumstances, we find that benign applications have significantly less impact on victim applications' HPCs compared to side-channel attack applications. Next, we present the problem of instance level false positive rate and demonstrate the importance of extracting customized features from HPCs traces intervals to achieve a higher detection accuracy and lower instance level false positive rate for ML-based SCA detectors. Then, the proposed real-time SCA detection methodology employs the customized features based ML classifiers by using only victim applications' microarchitectural information to enhance the SCAs detection accuracy and minimize the instance false positives rate while avoiding the corrupted, manipulated, or missing attackers' HPCs information.

The main contributions of this work can be summarized below:

- To eliminate the influence of missing attack profiling data or manipulation in the attack applications codes, this work proposes *SCARF* to detect SCA attacks in real-time using the minimal number of HPC features (only four features). The proposed approach detects SCAs based on differentiating HPCs data of only the victim applications under two conditions: 1) Victim under Attack (VA), and 2) Victim under No Attack (VNA).

- Various ML classification algorithms are explored to find the most accurate classifier for detecting side-channel attacks in real-time.

- The proposed ML-based detectors are trained using a customized set of HPC features to improve the detection accuracy further while lowering the false alarm rate.

- The False Alarm Minimization (FAM) method is proposed to reduce the instance level false positive rate with little latency.

3.1.1 Background and Motivation

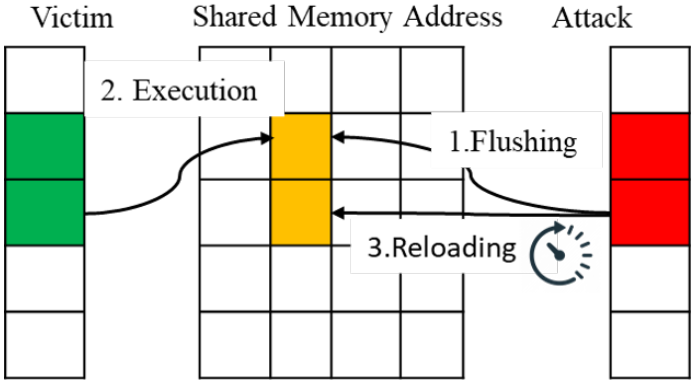
This section introduces the SCAs and time-series classification models used in this work. Then we highlight key motivations behind proposing *SCARF* framework for detecting side-channel attacks using low-level hardware features.

Side-channel Attacks

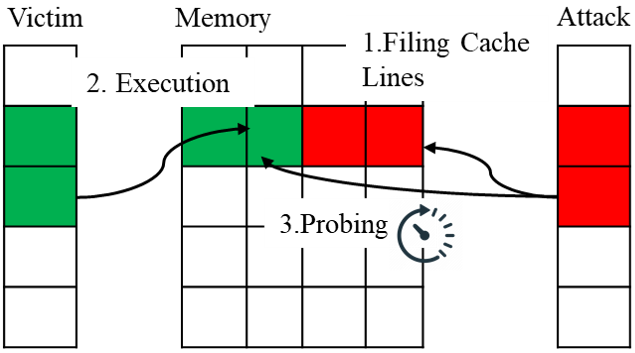
The emergence of different hardware components such as cache memory, branch predictor, etc. to enhance the modern microprocessors' performance, have led to the exposure of new hardware vulnerabilities in the systems. Such exposure makes a unique opportunity to spy on victim applications and infer sensitive information by side-channel attacks.

- *Flush+Reload* The research in [74,75] exploits the vulnerability of the page de-duplication technique by monitoring the memory access lines in the shared pages. This attack targets the Last-Level Cache in the CPU, flushes out victim applications' data in the cache, and waits for the victim application to execute, as shown in Figure 1-(a). After flushing the cache, the attacker attempts to access the data and measures the accessing time (latency). Shorter accessing time denotes that the victim application has accessed the data; otherwise, it has not been accessed.- *Prime+Probe* Without the memory de-duplication restriction, Prime+Probe [1] attack could be applied to a broad range of systems. This type of SCA consists of two different stages: Prime and Probe as shown in Figure 1-(b). In the Prime stage, the attacker builds the eviction sets, which are grouped cache sets causing potential conflict with victim applications, and then fills the cache with the eviction sets. Next, the attacker waits for the victim applications' execution and then re-accesses the eviction sets (Probe stage). If the accessing time is long enough, the victim application has accessed the data.

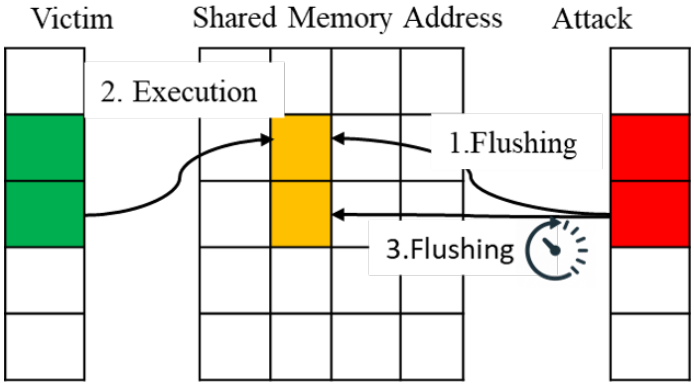
- *Flush+Flush* This SCA relies on the execution time of the flush instruction [3]. Unlike prior attacks, Flush-Flush does not make any memory access or rely on the data's access latency. The execution time of flush instruction depends on whether the data is stored in the cache. Flush-Flush uses the execution time of the subsequent flush instruction following the victim application's



(a) Flush+Reload



(b) Prime+Probe



(c) Flush+Flush

Figure 3.1: Working principle of three emerging cache-based side-channel attacks

execution, as shown in Figure 1-(c). The longer execution time of the flush instruction indicates that the corresponding data was brought to the cache and later was accessed by the victim application.

Time-series Classification

Since our work proposes a time-series classification based on HPCs data to detect the SCAs, we will introduce dynamic time warping and compare it with other time-series classification methods. There are several methods proposed for mining the time-series data including the dynamic time warping (DTW) [76], Shapelet [77, 78], Bag of Patterns(BOP) [79], Symbolic Aggregate approXimation (SAX) [80], etc. DTW calculates the distance between two time-series sequences and attempts to align two different sequences while giving the optimal one with the shortest distance. As a result, DTW achieves the measurement of the similarity between temporal sequences with different speeds. SAX is another time-series algorithm that first transforms data into the Piecewise Aggregate Approximation (PAA) representation, which will be symbolized into a sequence of discrete strings. Next, SAX deploys an approximate distance function that lower bounds the Euclidean distance. Shapelet algorithm introduces a new concept called "shapelet" a sub-sequence extracted from one of the time-series sequences. The Shapelet algorithm selects the Shapelet according to the ability of the Shapelet to classify time-series data.

Detection based on Victim Applications' HPCs Data

- *Unreliable Attackers' HPCs:* Prior studies on SCAs detection have mostly focused on profiling both victim and attack applications to collect hardware performance counters data for detecting whether an attack occurs or not [9, 16, 41]. Nevertheless, a recent work [72] presents the problem of detecting attackers by classifying attackers and benign applications based on HPC information with the aid of ML techniques. The work in [71] further points out the non-deterministic and over-counting problems of instructions associated with HPCs information, which the attackers can intentionally modify instructions slightly and manipulate the counters, hence thwarting such detectors.

- *SCAs Design Principle:* Current SCAs intentionally cause influence on victim applications' cache or branch predictor by flushing/priming cache, mistraining branch predictors and then observing accessing time of the cache sets, which changes caching victims' data and microarchitectural behaviors of victim applications [81]. This also provides the opportunity of detecting SCAs by

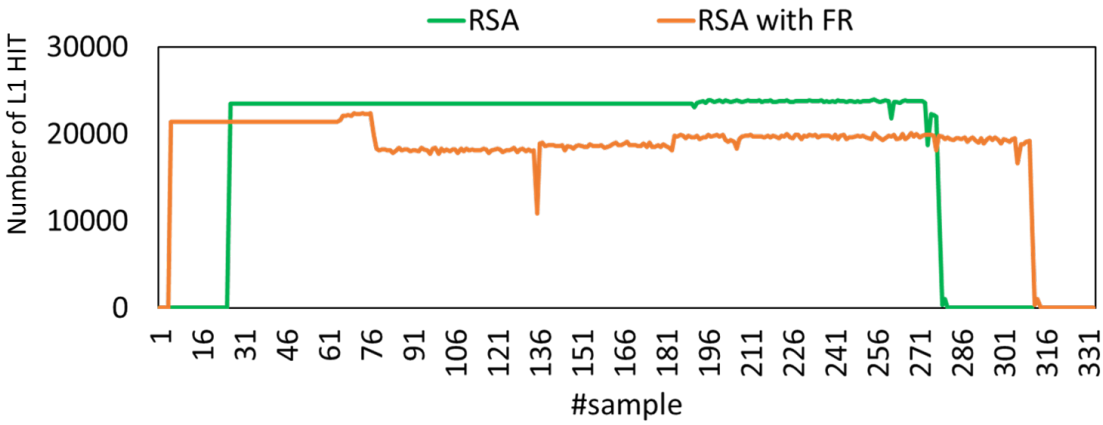


Figure 3.2: L1 HIT of RSA and RSA under Flush Reload attack

observing the alteration in microarchitectural behaviors. Furthermore, our experimental results, as shown in Figure 3.2 indicate that there exists a clear difference between the behavior of VNA and VA. In this motivational case study, the HPC traces of L1 HIT for the tested victim application (RSA) under no attack (RSA) and under L3 Flush Reload attack (RSA with FR) are illustrated. It can be observed that the L1 HIT of VA shows a significantly different trend compared to that of VNA. This observation clearly highlights the effectiveness of using HPCs data of only victim applications (excluding the impact of attack applications' HPCs) for detecting the behavior of SCAs.

Table 3.1: False Positive and False Negative evaluation (True: VNA; False: VA)

	Ground Truth	Predicted True	Predicted False
Interval Level	Actual True	True Positive	**False Positive**
	Actual False	**False Negative**	True Negative
Instance Level	Actual True	True Positive	**False Alarm**
	Actual False	**Missed Alarm**	True Alarm

False Alarm Problem

As depicted in Figure 3.3-(a), each run of a victim application is called an instance. For the purpose of real-time SCAs detection, a certain window size is used to decide the number of samples of each interval. Each instance could contain multiple intervals. In addition, Figure 3.3-(b) shows a

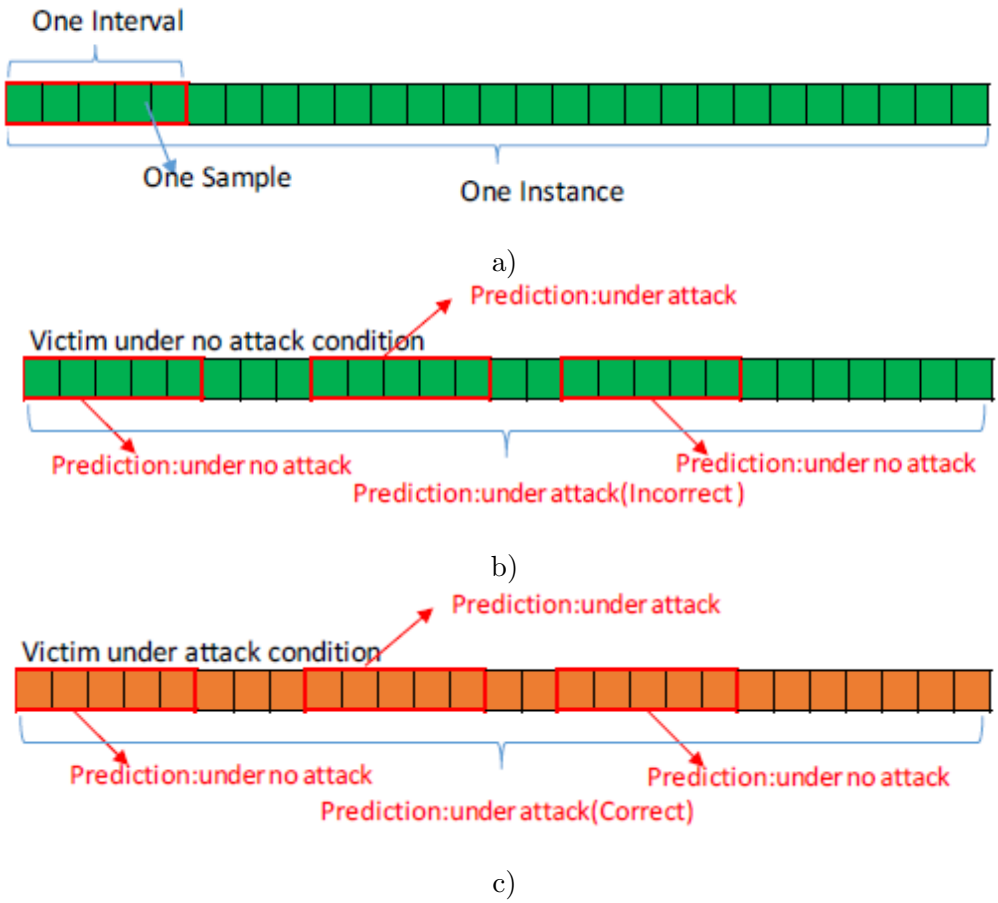


Figure 3.3: False Alarm Problem: a) Concept; b) VNA condition; c) VA condition

VNA instance divided into multiple intervals. In such cases, even if only one interval is predicted as VA by the ML-based detection technique, the whole instance will be classified incorrectly as VA. At the same time, Figure 3.3-(c) illustrates that the VA instance has two intervals classified as VNA and one interval classified as VA; the whole instance is still correctly classified as VA. Hence, even for prior works [9, 16] achieving high detection accuracy with low false positives, it is still hard to say that they can achieve a low instance level false alarm rate. To distinguish false positives of interval level and instance level, we deploy false alarm and missed alarm to represent false positive and false negative of instance level in the following sections as shown in Table 3.1.

Customized Features based ML Classifiers

Prior works capture the sum of HPCs value for a certain time period as features and employ traditional ML classification methods, achieving high prediction accuracy. They can cause a high false alarm rate as mentioned above. Hence, customized features based on ML classifiers

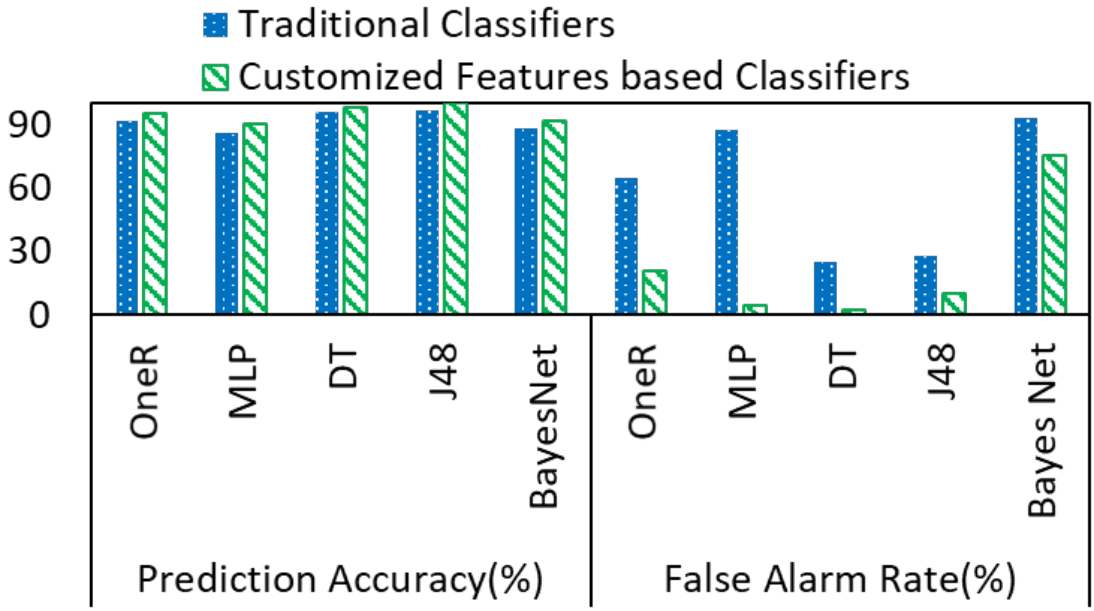


Figure 3.4: Traditional and customized features based classifiers comparison (datasets collected based on Section 3.1.2)

that extract more features, like min, max, stdev, and sum of an interval, are required to enhance the prediction accuracy further and reduce the false alarm rate. Prediction accuracy and false alarm rate of traditional and customized features based classifiers are plotted in Figure 3.4. It can be seen that customized features based ML classifiers outperform traditional ones by around 4% prediction accuracy. Furthermore, the false positive rate drops from 87.2% to 4.7% for MLP when applying customized features based classifiers. It can be concluded that extracting more features from HPCs time-series sequences can help ease the "under attack" bias mentioned in Section 3.1.1. Hence, real-time SCA detection needs customized features based classifiers with more extracted HPCs features which can help boost prediction accuracy and remarkably reduce false alarm rate.

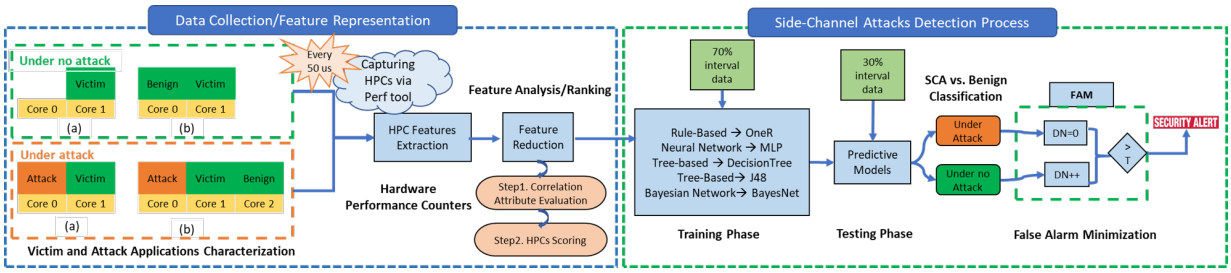


Figure 3.5: Overview of *SCARF*, the proposed real-time SCAs detection methodology based on victim application HPCs

Table 3.2: Architectural configurations

Processor	Intel I5-3470 CPU, single socket-4 cores
Frequency	1.2-2.6GHz
L1i Cache	32KB
L1d Cache	32KB
L2 Cache	256KB
L3 Cache	6144KB
Memory Capacity	8GB DDR3
Frequency	2600MHz

Table 3.3: The experimented victim and attack applications

Victim	Attack	Source
RSA	L3 Flush Reload	Masitk [74]
	L1 Prime Probe	Masitk
AES	Flush Reload	Xlate [75]
	L3 Flush Flush	Xlate
victim_function	Spectre	Spectre [4]

3.1.2 Proposed Methodology

In this section, we first present details of the experimental setup and configurations. And then the proposed *SCARF* methodology shown in Figure 3.5 will be introduced. As shown, *SCARF* is comprised of different steps such as data collection, feature reduction, training phase, testing phase, and false alarm minimization method. First, for feature extraction the "under no attack" and "under attack" HPC data will be collected within a) isolated scenario, and b) non-isolated scenario. The "isolated" environment refers to the case that a computer only processes victim applications; whereas the "non-isolated" environment denotes that a computer system processes victim applications on one core while benign applications are being executed on the rest of the cores. Then customized features are extracted, and the data will be used to train various classifiers.

Table 3.4: The collected HPC features and their ranking

Ranking	**HPC Name**	**Ranking**	**HPC Name**
1	L1 HIT	9	L1 MISSES
2	UOPS_RETIRED	10	BRANCHES MISPREDICTED
3	BR_NONTAKEN_CONDICTIONAL	11	L2 HIT
4	ALL BRANCHES RETIRED	12	TAKEN_INDIRECT_NEAR_CALL
5	INST_RETIRED_ANY	13	L3 HIT
6	L2 MISSES	14	ITLB_MISSES
7	BR_TAKEN_CONDITIONAL	15	DTLB_STORE_MISSES
8	L3 MISSES	16	DTLB_LOAD_MISSES

Next, the trained models will be employed in the testing phase, and the false alarm minimization technique further assists in reducing the false alarm rate.

Experimental Setup and Data Collection

In this work, all experiments are conducted on an Intel I5-3470 desktop with four cores, 8GB DRAM, and a three-level cache system. Victim applications and side-channel attacks are selected from Mastik [74] and Xlate [75]. Furthermore, MiBench [82] benchmark suite is used to represent benign applications. In this work, we propose using a customized tool to collect hardware performance counters based on Model-Specific Registers (MSRs). The proposed customized monitoring tool collects HPCs per processor at a microsecond scale with privileged access to avoid HPCs contamination from other processes addressing the overcounting challenges presented in a recent study [71]. Based on the behavior and functionality of studied SCAs, 16 HPC features are considered in this work for further analysis as listed in Table 3.4. These hardware performance counters data are collected using the four HPC registers in the experimented I5 processor every 50 microseconds. Each pair of a VNA and VA executes for 50 times. Next, both VA and VNA HPC data are merged together to create the final dataset. Furthermore, Weka data mining tool is employed to implement ML classifiers. To validate each of the utilized ML classifiers, a standard 70%-30% non-biased dataset split for training and testing is followed in which 70% of the randomized data (known applications) is used for training the classifiers, and the rest of 30% (unknown applications)

is used for testing evaluation.

Customized Features based Classifiers

The proposed customized features based classification is comprised of three main steps: 1) feature extraction and representation; 2) HPCs selection due to a limited number of registers for effective real-time detection of the attacks; 3) training the ML classification algorithms.

- *Features Vector Extraction and Representation* For the proposed classification, the raw data will be transformed from a time sub-sequence to a vector of features. In the first step of the transformation process, the raw data will be received from the monitoring module. The time-series sub-sequences properties will be extracted, which include statistics of distribution values (including max, min, stdev, and sum). In order to effectively determine the most prominent features for the purpose of real-time SCA detection, we deployed the Greedy Forward Selection algorithm [83, 84] and we found that max, min, StatAv, and sum contribute more to assisting in distinguishing the difference between "under no attack" and "under attack" conditions. Hence, the input for each transformation is $T = (t1, t2, ..., tm)$ where tm is a vector of HPCs values. Also, the outputs of transformation are a vector of actual HPC values i.e., L1HIT sum, L1 HIT max, etc.

- *HPCs Feature Reduction* Detecting side-channel attacks using ML models requires representing programs at low-level features which leads to a high-dimensional data processing involved large computational overheads and complexity. Furthermore, incorporating irrelevant features would lead to lower accuracy and performance for the classifiers. Hence, it is crucial to perform an effective feature reduction of collected data to alleviate unnecessary computational overheads and determine the most prominent low-level features [73]. In order to detect the SCAs at real-time with minimal overhead in *SCARF*, we intend to identify a minimal set of critical HPCs that are feasible to collect even on low-end processors with small number of HPCs in a single run. Therefore, a subset of HPC features is selected representing the most important features for classification. The selected features are then supplied to each ML-based SCA detector. The detector attempts to find a correlation between the feature values and the application behavior to predict the SCA.

Given the limited number of HPCs available in modern microprocessors (only 4 HPCs on tested Intel I5-3470) to be collected at one time simultaneously, it is necessary to identify the most

important HPCs for classifying the VA and VNA conditions for different types of SCAs [73]. For HPCs reduction, we employ Correlation Attribute Evaluation (*CorrelationAttributeEval* in Weka) with its default settings to calculate the Pearson correlation between attributes (HPC features) and class (VA and VNA conditions). Correlation attribute evaluation algorithm calculates the Pearson correlation coefficient between each attribute and class, as given below:

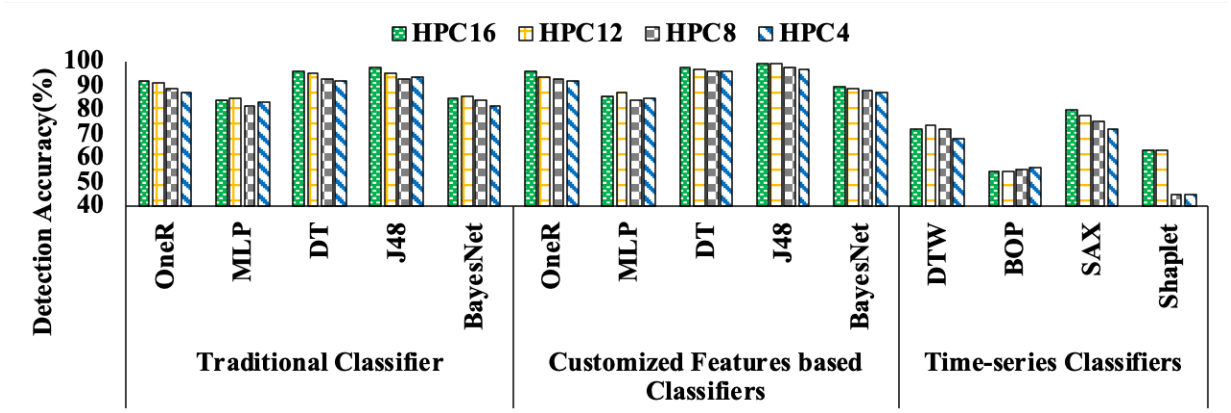
$$\rho(i) = \frac{cov(Z_i, C)}{\sqrt{var(Z_i)\ var(C)}} \quad i = 1, ..., 16 \tag{3.1}$$

where ρ is the Pearson correlation coefficient. Z_i is the input dataset of event i ($i = 1, \ldots, 16$). C is the output dataset containing labels, i.e. "Under Attack" or "Under No Attack" in our case. The $cov(Z_i, C)$ measures the covariance between input data and output data. The $var(Z_i)$ and $var(C)$ measure variance of both input and output datasets, respectively. Next, the sum score of each HPC features (min, max, stdev, and sum in this work) will be calculated and HPCs will be ranked according to sum score as shown in Table 3.4.

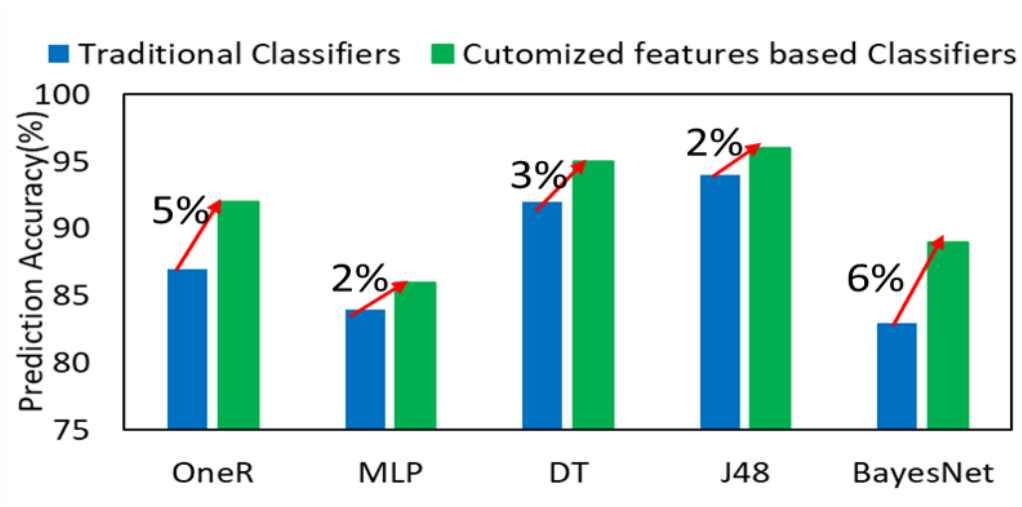
- *ML Classifiers Implementation* For the purpose of a thorough analysis of various types of ML classifiers, OneR, MLP (multilayer perceptron), DT (decision table), J48, and BayesNet ML algorithms are deployed as our final classification models. The rationale for selecting these machine learning models are: firstly, they are from different branches of ML: regression, neural network, decision tree, and rule-based techniques covering a diverse range of learning algorithms which are inclusive to model both linear and nonlinear problems; secondly, the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the SCA detection problem in our work. As mentioned before, only four HPCs can be collected for most processors at once due to a limited number of registers for storing them. Hence, reducing the number pf HPCs required for ML models is important to eliminate the need of multiple runs. For this purpose, various number of HPCs from 16 to 4 (16, 12, 8 and 4 selected based on the ranking in Table 3.4) are examined to evaluate the influence of reduced HPCs on classification accuracy and highlight the importance and motivation of using a lower number of HPCs (only 4) for effective real-time SCA detection in *SCARF*.

False Alarm Minimization (FAM)

Previous real-time SCAs detection methods are biased to under attack category that results in an increasing the false alarm rate. Hence, to address this challenge, we propose the False



a)



b)

Figure 3.6: Prediction accuracy comparison: a) prediction accuracy of proposed customized features based classifiers and the rest two type classifiers; b) zooming in prediction accuracy of traditional and customized features based classifiers

Alarm Minimization (FAM) technique in which we delay the "under attack" detection decision until receiving a certain number of continuous intervals, delay number (DN), before reporting it as "under attack," while minimizing detection latency.

To this aim, we assume false positives are evenly distributed among each instance, which results in the highest false alarm rate with the same false positive rate. Following, the value setting of DN will be demonstrated. As mentioned above that the evenly distributed false positives lead to the highest potential false alarm rate, we propose the method of setting DN value to ensure the highest potential false alarm rate with known false positive rate and the number of instance

intervals that can be obtained after testing classification models. First, we suppose $DN = m$, the number of $intervals = n$, false positive $rate = s$ and acceptable false alarm rate is t. There are $n - m + 1$ possible cases of m consecutive intervals that are incorrectly identified as "under attack". Therefore, the false alarm rate can be calculated by $FAR = (n - m + 1) * (s\%)^m < t$. Since n, s, and t are known, the minimum DN value can be deduced according to the equation.

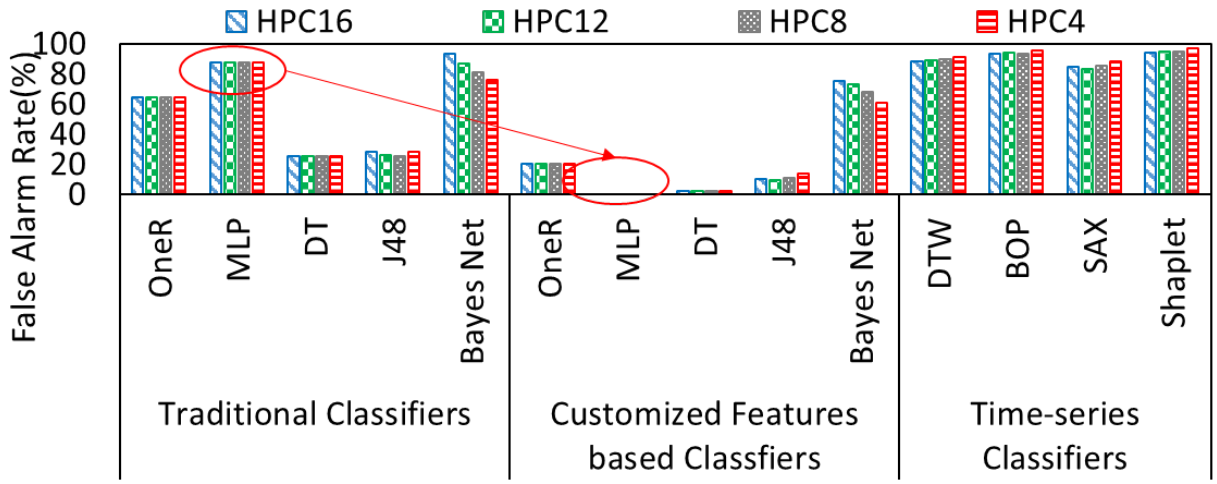


Figure 3.7: False alarm rate comparison

3.1.3 Results Evaluation

In this section, we evaluate the effectiveness of *SCARF* detection approach and compare it in terms of various evaluation metrics including detection accuracy and false alarm rate of proposed customized features based classifiers over traditional and time-series classifiers. Lastly, we evaluate the FAM influence on minimizing false alarm rate and reduction of attack detection rate.

ML Classifiers Comparison

In this work, time-series classification techniques are adopted for further comparison. To this aim, four prominent time-series classifiers, including Dynamic Time Warping (DTW), Bag-Of-Patterns (BOP), Symbolic Aggregate approXimation (SAX), and Shapelet are deployed that represent various categories of time series classification techniques including shape-based, structure-based without/with order information, and sub-phase shape-based categories. In order to present the effectiveness of using customized features, ML classifiers are fed with only sum value of HPCs named as traditional classifiers. The implemented ML classification algorithms are OneR,

multi-layer perceptron (MLP), DecisionTable (DT), J48, and BayesNet that are covering a diverse range of ML techniques.

- *Detection Accuracy:* Figure 3.6-(a) presents the SCA detection accuracy with a varied number of HPCs for the proposed *SCARF* (customized features based classifiers) and existing works (traditional and time-series classifiers using different techniques) [9, 16]. One can observe that the time-series classifiers achieve a lower accuracy despite utilizing more number of HPC features, i.e., the SCA detection accuracy is $< 70\%$ on average. Hence, existing time-series classifiers are not the optimal solution for real-time SCA detection. By comparison, the proposed and traditional classifiers achieve above 80% prediction accuracy despite utilizing less number of HPCs, which makes them formidable candidates to consider for real-time SCA detection. Figure 3.6-(b) zooms in the comparison between proposed and traditional classifiers. It can be seen that *SCARF* method by using customized features based ML classifiers is able to further boost prediction accuracy, ranging from 2% to 6%.

- *False Alarm Rate:* As discussed, despite a high detection accuracy, one of the major challenges associated with detection is the false alarms in which we evaluate the false alarm rate for different techniques below. Figure 3.7 depicts the false alarm rate with proposed and existing techniques when utilizing a varied number of HPCs for SCA detection. The false alarm rates produced by traditional classifiers based SCA detection are significantly high, 57% on average across all ML techniques and HPC values. This is due to the fact that traditional methods are biased to "under attack". However, the proposed *SCARF* technique with using customized features employs more features that aid the ML classifiers in predicting "under attack" scenario with higher confidence and accuracy. Taking MLP-based SCA detector as an example, the proposed customized classifier can decrease false alarm rate from 87% (obtained when utilizing traditional classifier) to 4.7%, though the detection accuracies are similar. Furthermore, time-series classifiers have shown above 80% false alarm rate.

Evaluation of FAM Technique

In this section, we evaluate the attack detection accuracy and false alarm rate of customized features based classifiers with the usage of proposed FAM. As described in Section 3.1.2, setting DN value is based on interval level positive rate, the number of instance intervals, and acceptable false alarm rate. Thus, the DN is set to 2/4 to ensure that false alarm rate is below 30%, 5%.

Furthermore, they are compared with DN=1 which corresponds to no "under attack" delay. It can

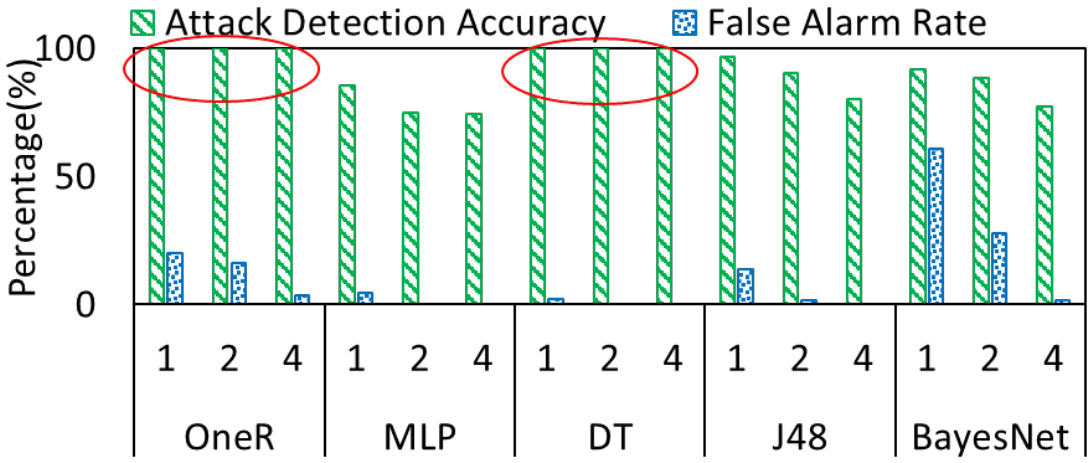


Figure 3.8: Attack detection accuracy vs false alarm rate with various DN values

be seen that increasing the number of DN from 1 to 4 does not have any impact on the attack detection accuracy of OneR and DT classifiers (remains 100%). In the meantime, the false alarm rates of the two classifiers decrease to 3.6% with DN=4 for OneR and 0% with DN=2 for DT classifier. As can be observed for the rest of three ML classifiers, the reduction of false alarm rates can be achieved at the cost of lowering the attack detection accuracy. For instance, in J48 classifier, false alarm rate decreases from 13.6% to 1.7% and to 0%, respectively, while the attack detection accuracy decreases from 96.7% to 90% and then to 80.5% with DN increasing from 1 to 4. It can be concluded that by deploying the proposed FAM technique the false alarm rate can be effectively reduced, while it may cause relative detection accuracy loss for some classification techniques.

3.2 Hybrid Dynamic Time Warping and Gaussian Distribution Model for Detecting Emerging Zero-Day Microarchitectural Side-Channel Attacks

To address the SCAs issues, some prior works [23, 25, 36, 85] propose mitigation methods to alleviate the influence of side-channel attacks. However, such solutions either require extra hardware design and/or are only effective to a subset of SCAs, ignoring the impact of zero-day attacks. Other studies [9, 16, 17, 19, 31, 41, 86–88] propose the use of microarchitectural pattern analysis captured through Hardware Performance Counters (HPCs) to detect the existence of side-channel attacks. For

instance, [16] offers an HPC monitoring model of both victim and attack applications to detect the SCAs. Based on the obtained HPCs, the HPC events of victims' and attacks' traces are correlated. Similarly, in [9] the authors present CloudRadar framework which aims at detecting cross-VM SCAs by making use of HPC patterns. This work comprises two phases: a) identifying sensitive applications by comparing offline generated signatures based on HPCs, and b) correlating victim HPCs and attack HPCs.

While there has been much progress in terms of identifying various SCAs in the last few years, there are still two major challenges involved with existing SCAs detectors. First, the fast-paced development of the emerging SCAs to circumvent the current detection techniques has not been properly addressed. Hence, the existing methods are not able to detect the unknown (zero-day) attacks. Secondly, existing works on SCAs detection have primarily focused on one or a few ML techniques for attack detection [9, 16, 17]. Such an analysis leaves a void in terms of performance of attack detection, as various ML classifiers yield different performance in detecting various types of attacks [73, 87, 89].

In order to solve the aforementioned challenges, in this work we propose *HybriDG*, a hybrid Dynamic Time Warping (DTW) and Gaussian distribution model to accurately detect both known and unknown emerging side-channel attacks at runtime. *HybriDG* employs DTW time-series classification to calculate the similarities of victim under no attack (VNA) and victim under attack (VA) HPCs traces and then apply Gaussian distribution to set a threshold of the similarities based on the optimal false alarm rate. During the testing part, only victim HPCs features are collected and fed to the dynamic time warping model. Lastly, the distance from DTW is compared with the threshold decided by Gaussian distribution to detect if the victim application is under attack or under no attack condition. *HybriDG* shows a significantly high detection accuracy for both known and unknown attacks with a low false positive rate eliminating the need to collect the attacks' HPCs data for training the SCAs detector.

3.2.1 Hardware Performance Counters (HPCs) Data

Hardware performance counters are special-purpose registers built-in modern microprocessors that count hardware-related events such as instructions executed, cache misses suffered, or branches mispredicted [73, 88, 90]. HPCs data are extensively used to auto-tune/profile appli-

Table 3.5: Selected Monitoring HPCs List

L1 HIT	L1 MISSES
L2 HIT	L2 MISSES
L3 HIT	L3 MISSES
All BRANCHES RETIRED	BRANCHES MISPREDICTED

cations [91], analyze and optimize performance [92], malware detection [73, 93] and SCAs detection [31, 87, 94].

This work only employs victim applications' HPCs and collects HPCs data by separate processing cores instead of counting them by application's thread. For this purpose, a PAPI-based HPCs monitoring tool is employed, and one HPC sample is collected every 50 microseconds. We have collected 8 HPC events listed in Table 3.5 to train our detector. These HPCs are selected based on the attacks' design methodology influencing the cache and branch predictor units. Both HPCs of victim under no attack and victim under attack are needed to collect. Hence, victim applications are first executed solely to obtain victim under no attack data. And then, victim and attack applications are executed concurrently to build victim under attack dataset.

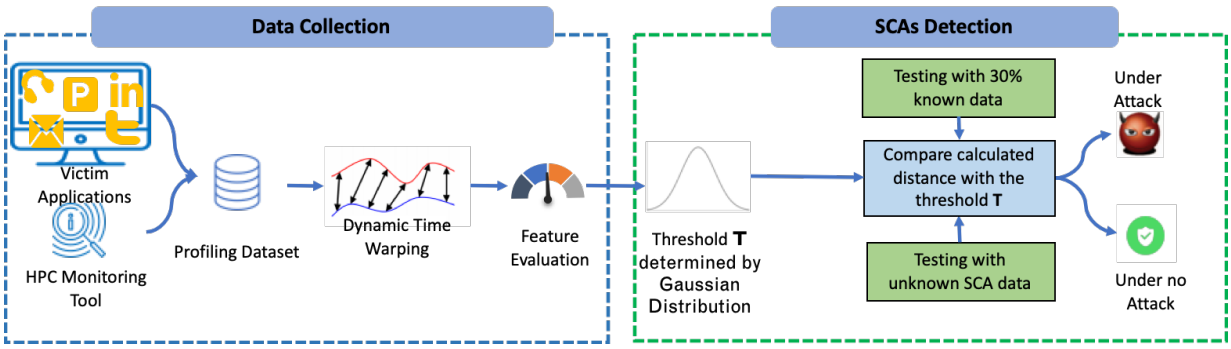


Figure 3.9: Overview of *HybriDG*, the proposed hybrid Model for detecting emerging zero-day SCAs

3.2.2 Proposed Methodology

In this section, we present the details of our proposed *HybriDG* model to detect the known and unknown side-channel attacks. Figure 3.9 depicts an overview of the proposed *HybriDG*. As seen, the proposed *HybriDG* detector contains two major parts: a) offline data collection and HPC features evaluation, b) distance threshold determination (T) with dynamic time warping and

Gaussian distribution, and online prediction with a threshold (T) that each part is discussed in details below.

Threat Model

The proposed *HybriDG* is designed to secure multi-core computing processors against SCAs that employ inclusive and non-inclusive caches on the Intel processor architectures, in which malicious and benign processes use shared libraries. In addition, it considers that the applications are executing in a Linux-based single OS environment, where both victims and attacks reside in the physical machine, on different processing cores. The system can face different SCAs, including shared-memory based SCAs such as Flush+Flush, Flush+Reload, and none shared-memory based SCAs such as Prime+Probe. In order to create the known and unknown threat conditions, two out of the three considered SCAs are used as known attacks, and the third attack is used for modeling unknown attacks. As a result, the known attacks can be profiled, and the corresponding runtime HPCs information is stored in database for training ML models. Moreover, the unknown attack is deployed to test the effectiveness of ML-based detectors built in the proposed *HybriDG* in detecting zero-day SCAs.

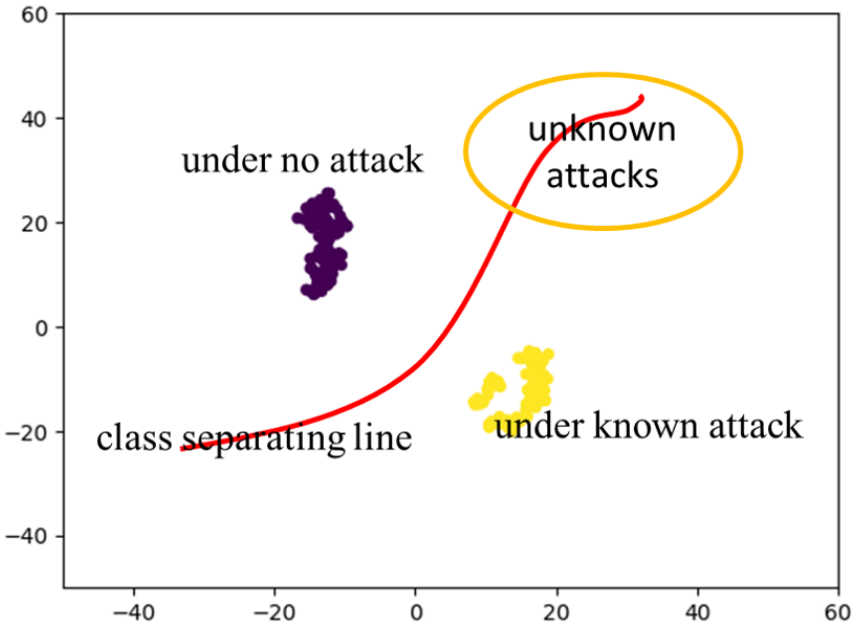


Figure 3.10: TSNE plot for victim under no attack and victim under known attacks samples

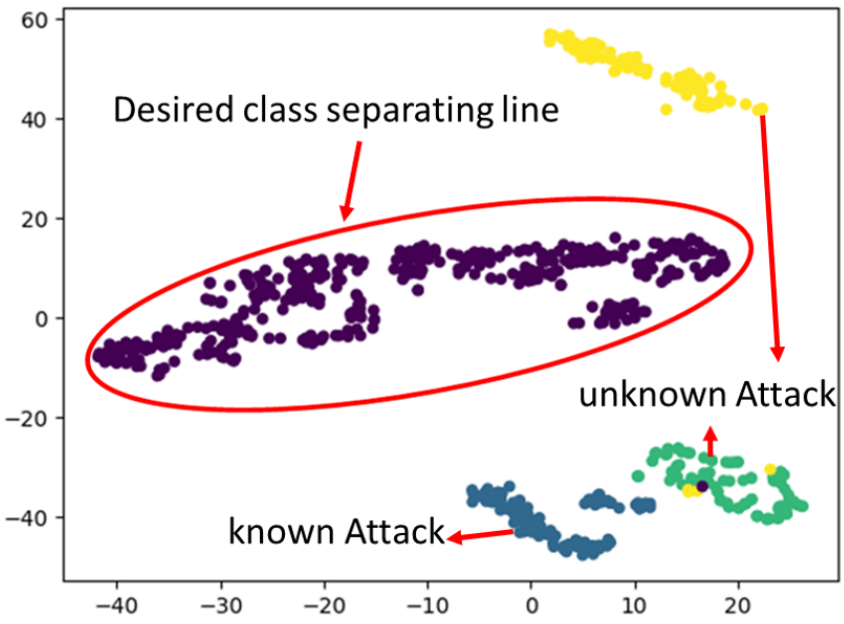


Figure 3.11: TSNE plot with desired classifying line for victim under no attack, known attack, and unknown attack samples

Classifying Unknown Dataset

As shown in Figure 3.10, victim under no attack and under known attacks temporal sequences are plotted using TSNE algorithm. It can be observed that samples can be easily separated under no attack and under known attacks. In addition, to conduct binary classification, prior classification models which are trained with under no attacks and under known attacks dataset could construct a line shown in Figure 3.10, which separates samples into two classes. However, unknown attacks might locate on both sides of the line, which results in the misleading and accuracy degradation of the ML classifiers. As a result, to achieve a high detection accuracy, we need to construct a classification line, as shown in Figure 3.11, which defines the threshold of "under no attack" and any samples outside the line is classified as "under attack".

Dynamic Time Warping (DTW)

DTW was introduced into classification problems and time-series mining by Berndt and Clifford [95], where dynamic programming was used to align sequences with different lengths. The main idea of DTW is to find an optimal match between two sequences by allowing a nonlinear mapping of one sequence to the other sequence and minimizing the distance between two sequences. Considering two HPC temporal sequences A and B as follows: A(a1, a2,..., an) and B(b1, b2,..., bm), DTW finds an optimal warping path between A and B by using dynamic programming to calculate the minimum cumulative distance γ (n, m), where γ (i, j) is defined in Equation 3.2:

$$\gamma(i,j) = (ai, bj)^2 + \min \begin{cases} \gamma(i-1,j) \\ \gamma(i-1,j-1) \\ \gamma(i,j-1) \end{cases} \tag{3.2}$$

This allows DTW to practically match similar shape sequences together, even though the sequences may be shifted or out of phase, and this qualification has further made DTW viable for a multitude of time-series classification problems. Keogh and Pazzani [96] considered replacing the value of each data point with the first derivative within the dynamic time warping distance function to resolve such a problem.

In this work, DTW is employed to calculate the similarities among victim under no attack HPC sequences and among victim under no attack and attack HPC sequences. For the following sections, the two distances calculated when victim application are under no attack and attack are abbreviated as VNAD (Victim under No Attack Distance) and VAD (Victim under Attack Distances).

HPCs Evaluation

As mentioned, a threshold of distances calculated by DTW needs to be decided. As shown in Figure 3.12, if a large threshold value is selected, some attacks could potentially bypass the detector. Whereas, if the threshold is too small, then some "under no attack" samples could be classified as "under attack", leading to a higher number of false positives in the classification result. Hence, the goal here is to carefully set a confident threshold that not only could divide VA and VNA sequences, but also is strict enough for detecting further unknown attacks. To this aim, we should choose the HPC features whose temporal sequences' distances determined by DTW are distributed

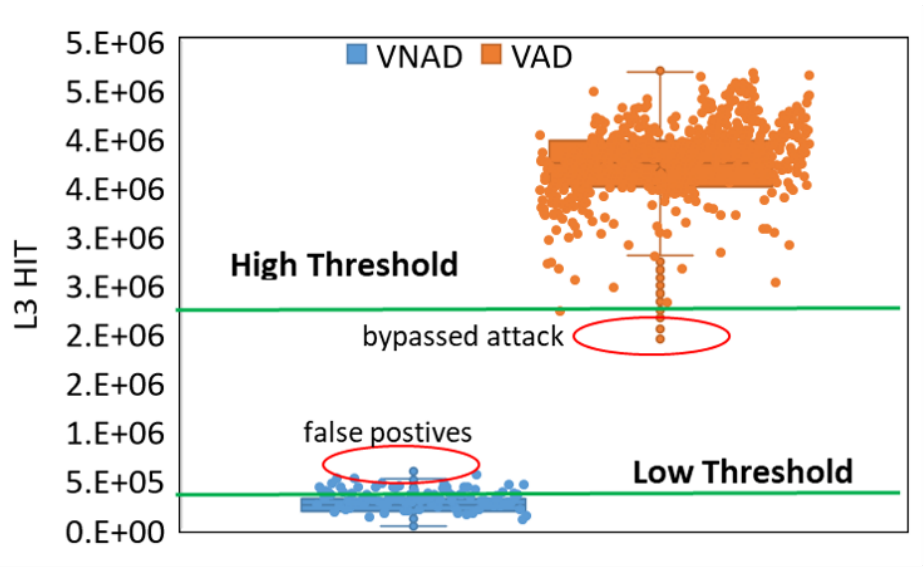


Figure 3.12: Different threshold influences(VNAD: victim under no attack distances; VAD: victim under attack distances.)

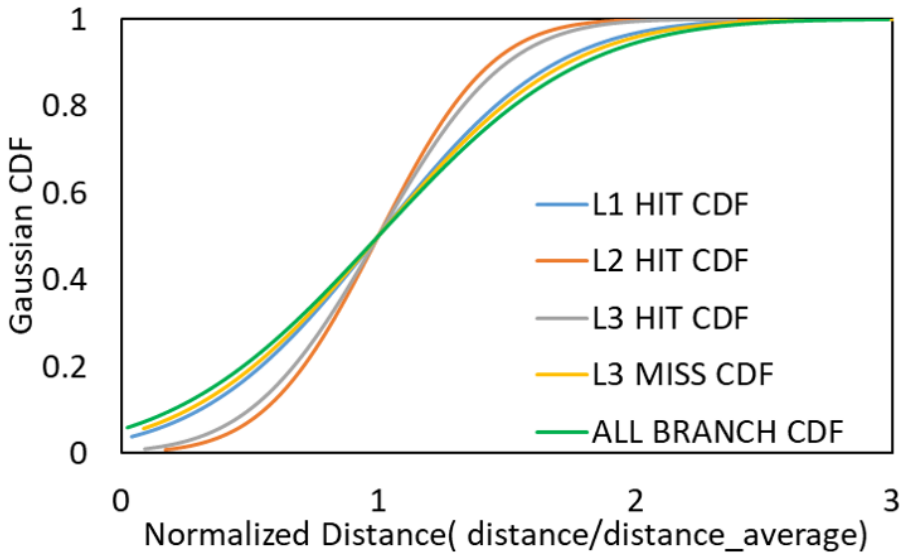


Figure 3.13: Gaussian distribution of various HPCs temporal traces

densely with the smallest possible variance. For this purpose, the CDF (cumulative distribution) of Gaussian distribution is used to plot the distances of HPCs sequences, as shown in Figure 3.13. Each HPC distances are normalized with the HPC's average distance to eliminate the influence of different HPCs value ranges. The Figure 3.13 shows that the L2 HIT feature value converges to 1 faster than other features indicating that by setting the same threshold, the L2 HIT can achieve higher VNA predicted correctly as "under no attack" compare to other HPC features. Table 3.6

further shows the theoretical false positives when using 1.5 as a threshold across different HPC features. It can be found that the L2 HIT could achieve higher VNA detection accuracy and a lower false positive rate. Therefore, we select the L2 HIT feature for building the classifiers according to the methodology mentioned above in this work.

Table 3.6: VNA dection accuracy and false positive rate with the 1.5 × distance_average across various HPCs

HPCs	VNA Accuracy	False Positive Rate
L1 HIT	82.2%	17.8%
L2 HIT	**92.6%**	**7.4%**
L3 HIT	89.9%	9.0%
L3 MISS	80.7%	19.3%
BRANCHES	79.0%	21.0%

Table 3.7: Theoretical false positive rate and corresponding L2 threshold value

Threshold Setting	Theoretical False Positive Rate	L2 Threshold
μ+σ	1/10	263245
μ+2σ	1/100	330946
μ+3σ	1/1000	398647
μ+4σ	1/10000	398647
μ+5σ	1/100000	398647
μ+6σ	1/1000000	601752

Gaussian Process for Threshold Determination

After choosing the most suitable HPC feature, the threshold needs to be set. The Gaussian distribution of HPCs temporal traces' distance value can estimate the percentage of points with a larger distance value than a certain threshold, which is a false positive rate. For example, the percentage of points with a value larger than $(\mu+\sigma)$ is 10%. Hence, the theoretical false positive rate is 10% when the threshold is $(\mu+\sigma)$ according to equation 3.3. We also study different thresholds

that influence the final prediction result in this work. The details of different thresholds are listed in Table 3.7 based on the HPC choice of L2 HIT. As shown in Figure 3.12, the smaller the threshold is, the higher the theoretical false positive rate is, and the larger the threshold is, the higher the possibility of missing "under attack" detection is. Therefore, choosing an optimal value to meet the false positive rate requirement and maintain high "under attack" detection accuracy at the same time is crucial. In this work, we choose $(\mu + 3\sigma)$ as a threshold to evaluate the proposed approach, meaning that the theoretical false positive rate is considered as 0.001.

$$f(x) = \frac{1}{\sqrt{(2\pi\sigma)}} * e^{\frac{(x-\mu)}{2\sigma^2}} \tag{3.3}$$

Table 3.8: Selected classifiers

Machine Learning Category	Classification Model
Traditional Classifier	J48, SGD, OneR
Time-series Classifier	1NN-DTW, BOPF, Shapelet
Deep Learning	LSTM, FCN, LSTM-FCN, ALSTM-FCN
Proposed Classifier	*HybriDG*

3.2.3 Experimental Results and Evaluation

In order to comprehensively evaluate the effectiveness of the proposed detector, various classification algorithms from three main categories of ML are selected and compared with *HybriDG* in terms of detection accuracy for known and unknown attacks, detection latency, and efficiency analysis. The selected ML classifiers are listed in Table 3.8. As shown, for the thorough analysis of the suitability of standard ML techniques for zero-day SCAs, we have implemented different types of classifiers, including traditional classifiers (tree-based J48, support vector machine based SGD, rule-based OneR), Deep learning classifiers (LSTM, FCN, LSTM-FCN, ALSTM-FCN), and Time-series classifier (DTW, BOPF, Shapelet).

Experimental Setup

The experiments are conducted on an Intel I5-3470 desktop with four cores and 8GB DRAM, a three-level cache system. In this on-chip cache memory subsystem, while L1 and L2

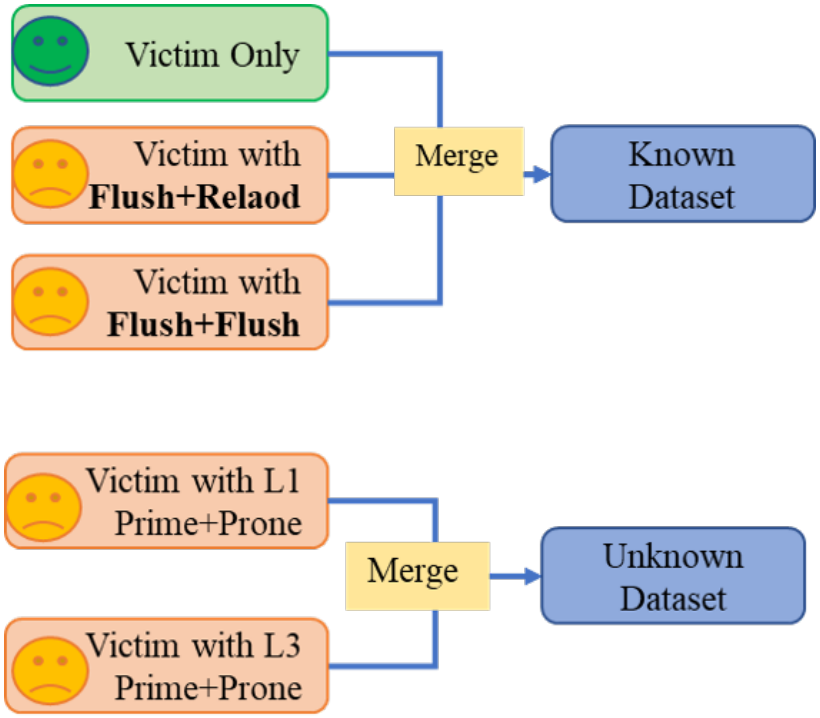


Figure 3.14: Testing datasets

caches are exclusively separated, L3 cache memory is inclusive and shared among all cores meaning that flushing out the data in the L3 cache will also remove the data in L1, providing vulnerability for LLC cache attacks to be exploited. In addition, the operating system is Ubuntu 16.0.4 LST with Linux kernel 4.13. In order to perform our experiments, AES and RSA are used as victim applications, and Flush+Reload, Flush+Flush, and Prime+Probe are used as attacks. In order to evaluate the effectiveness of detecting unknown attacks, Flush+Reload and Flush+Reload are used as known attacks, and the remaining applications are used as unknown attacks that are not included in the training dataset.

Testing Dataset

As introduced in Section 3.2.2, victim under no attack and under attack sequences are fed into the DTW model, and a threshold determines whether the victim is under attack or not. All SCA applications experimented in this work are divided into known attacks and unknown attack sets to model real-world application scenarios. To this aim, as illustrated in Figure 3.14, victim only and victim with known attacks form the known dataset, and 70% of the known dataset is then employed as a training dataset to choose the most effective HPC and determine the threshold. And

the remaining 30% of the reference dataset is used as a testing dataset to model unknown zero-day attacks. Therefore, the testing results are presented regarding detection accuracy for the known and unknown datasets.

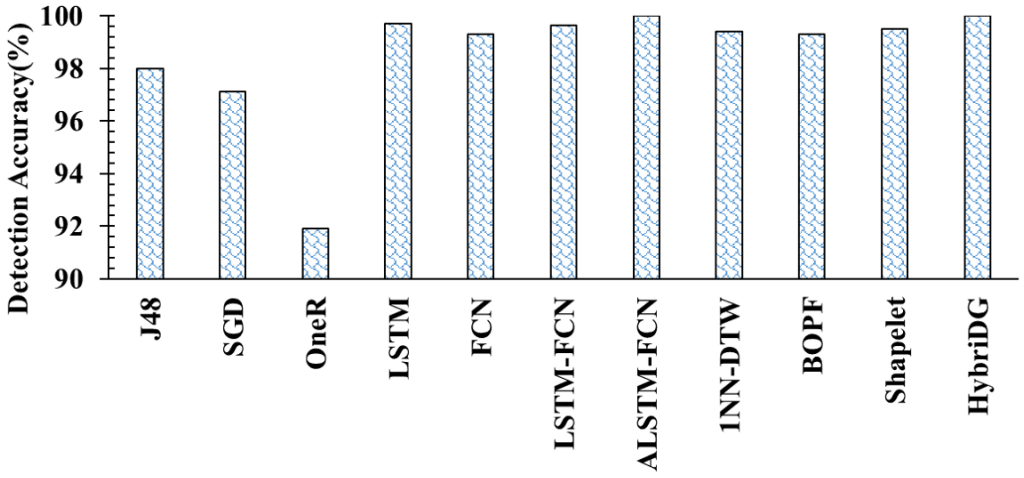


Figure 3.15: Known attacks detection accuracy of various classifiers

Known Dataset Testing Results

- *Detection Accuracy* Figure 3.15 presents the detection accuracy (rate of the correctly classified samples) of all evaluated and proposed classifiers. It can be observed that traditional classifiers, deep learning-based classifiers, and time-series based classifiers can achieve above 90% detection accuracy for the known dataset. This observation validates the plot given by TSNE algorithm in Figure 3.10, which shows that "under no attack" and "under known attacks" samples are located separately and can be easily classified by means of standard ML algorithms. The distribution in Figure 3.10 demonstrates that even by applying a simple classifier like rule-based OneR we could achieve 92% detection accuracy with only 1 HPC feature. It is also noticeable that a more complex deep learning and timer-series based classifiers outperform the traditional classifiers, achieving above 99% accuracy. Among all implemented ML-sbased detectors, the proposed detector *HybriDG* obtains the highest detection accuracy of 100%.

- *Classification Robustness* Receiver Operating Characteristics (ROC) Curve is produced by plotting the fraction of true positives rate versus the fraction of false positives for a binary classifier. The best possible classifier would thus yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives. The Area under the ROC

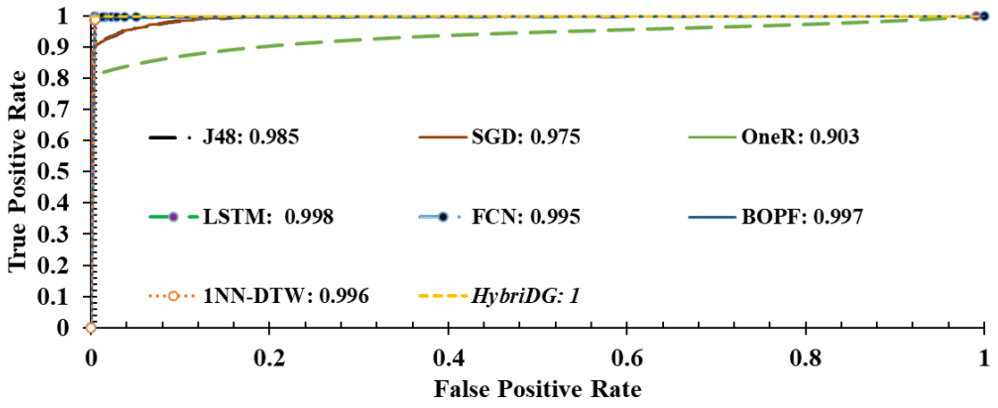


Figure 3.16: ROC Curve and AUC value of various classifiers

Curve (AUC) metric corresponds to the probability of correctly identifying "under attack" and "under no attack" and robustness is referred to how well the classifier distinguishes between the two classes, for all possible threshold values. Higher AUC indicates better robustness for ML classifiers. Due to space limitation, in Figure 3.16 we show the results for two classifiers from deep learning and time-series categories and compare them with traditional and the proposed *HybriDG* detector using the ROC Curve and corresponding AUC values. Similar to detection accuracy, it can be observed in Figure 3.16 that the *HybriDG* outperforms all other classifiers in terms of AUC, having the highest AUC and smallest distance to point (0,1) in ROC curve. Both deep learning and time-series classifiers yield to high robustness as well with slightly lower AUC value. It is also notable among three traditional classifiers, the rule-based OneR model delivers the lowest AUC value.

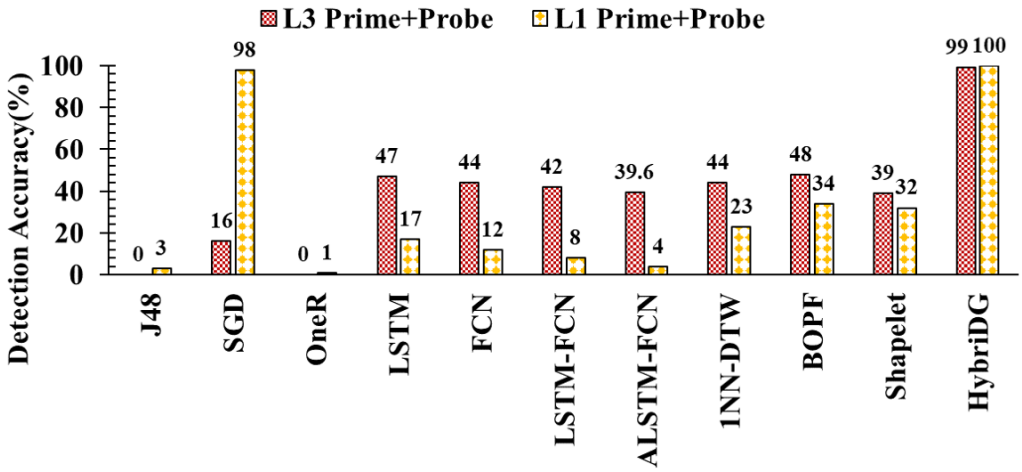


Figure 3.17: Unknown attacks detection accuracy

Unknown Dataset Testing Results

The unknown dataset contains victim under unknown attacks only (L3/L1 Prime+Probe) and is used for testing if the proposed SCAs detector is able to accurately detect zero-day attacks or not. As shown in Figure 3.17, L3 Prime+Probe and L1 Prime+Probe are unknown attacks, and the detection accuracy for each of them is demonstrated. As seen, the proposed *HybriDG* is substantially outperforming all other ML classification models achieving 100% for L1 Prime+Probe and 99% accuracy for L3 Prime+Probe detection. Moreover, one could observe that most of the experimented ML classifiers are not able to detect unknown attacks with high accuracy, delivering less than 50% detection accuracy, except SGD (77.8% and 98% for L3 Prime+Probe and L1 Prime+Probe, respectively). Though SGD shows better detection performance compared to the rest, it is still significantly less accurate and robust than the *HybriDG* detector highlighting the effectiveness of our proposed hybrid model for detecting emerging zero-day SCAs.

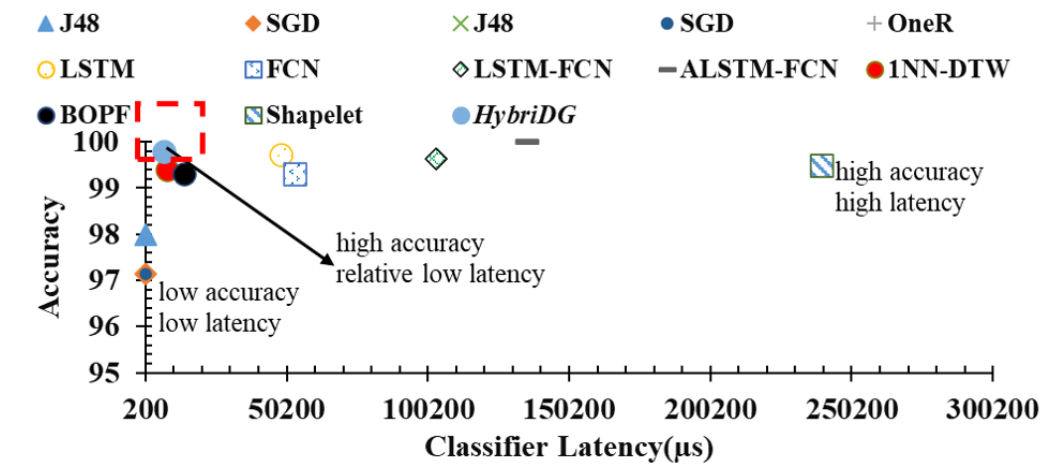


Figure 3.18: Efficiency comparison among various classifiers for known dataset

Efficiency Analysis: Accuracy vs. Latency

Figure 3.18 and Figure 3.19 present the trade-off analysis between detection accuracy and detection latency (computational costs) for known and unknown datasets, respectively. In Figure 3.19, accuracy is the average detection accuracy of L1 Prime+Probe and L3 Prime+Probe. It can be found that the proposed *HybriDG* achieves high detection accuracy (100% and 99.5%) with relatively low latency compared to deep learning or time-series classifiers for both known and

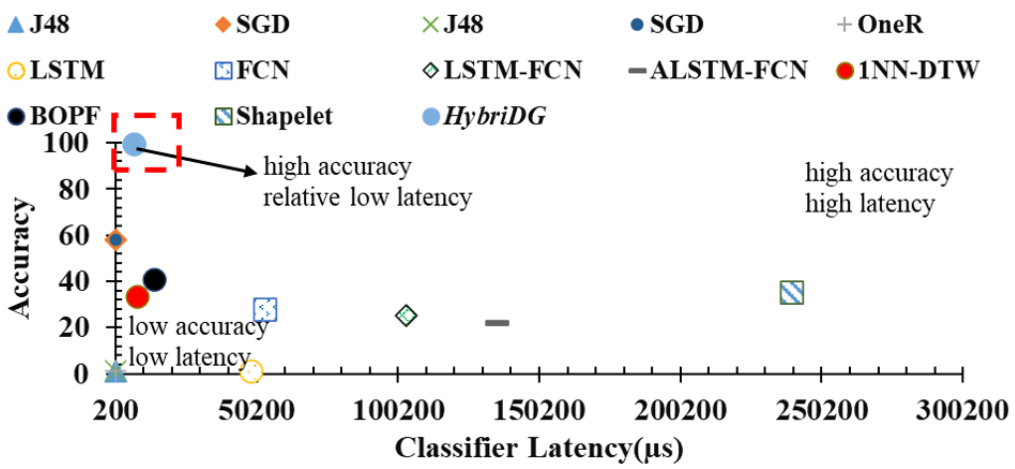


Figure 3.19: Efficiency comparison among various classifiers for unknown dataset

unknown datasets. Although traditional classifiers, like SGD can also achieve high accuracy (around 98%) for known dataset with lowest latency, its accuracy for unknown dataset (zero-day SCAs) is significantly low (only 57%), which still exposes computer systems to potential new attacks and security exploits. Also, while deep learning and time-series classifiers achieve high detection accuracy for the known dataset, they lack the ability to detect unknown SCAs with high accuracy and further result in higher latency than the *HybriDG* detector.

3.3 Conclusion

This chapter investigates the feasibility of using ML and the processors' hardware performance counters to design effective SCAs detectors to address the real-time capturing and zero-day hunting challenges. To achieve this, this chapter solves the challenge of the lack of attacks applications' HPCs data by analyzing the difference between Victim under Attack (VA) and Victim Under No Attack (VNA) conditions. Our comprehensive analysis indicates that HPCs data of VNA and VA show significantly different behavior providing the opportunity to detect SCAs with only victim applications' HPCs data. To build a real-time detector, we use HPCs importance evaluation with Correlation Attribute Evaluation algorithm to identify the most prominent HPC features. Furthermore, *SCARF* is further customized with different ML classifiers trained specialized set of features and False Alarm Minimization (FAM) technique to enhance the accuracy of SCA detection and reduce false alarm rate, respectively. In the second work of this chapter, we propose a hybrid

Dynamic Time Warping (DTW) and Gaussian distribution model called *HybriDG* to accurately detect both known and unknown emerging microarchitectural side-channel attacks at runtime. The proposed methodology calculates the similarities between victim HPCs traces collected from two conditions: victim under attack and victim under no attack, which is indicated by distances from the DTW model. Then, according to the distribution of victims under no attacks and victims under known attacks, only one HPC feature is selected, and the optimal threshold is determined according to the estimated false positive rate. After setting the threshold, the HPC temporal sequences will be fed into the DTW model to calculate the distance, which is compared with the threshold to identify whether a victim is under attack or not. Our comprehensive evaluation indicates that *HybriDG* achieves 100% detection accuracy for known attacks and 99.5% detection accuracy for unknown attacks using the most prominent microarchitectural feature (L2 HIT) outperforming standard ML models by up to 80% for unknown and 8% known attack detection.

Chapter 4

Hardware-assisted On-device Detection on Emerging Edge devices

Emerging embedded systems and Internet-of-Things (IoT) devices, which account for a wide range of applications are often highly resource-constrained that are challenging the software-based methods traditionally adopted for detecting and containing cyber-attacks (e.g., malware) in general-purpose computing systems. In addition to the complexity and cost (computing and storage), the software-based detection methods mainly rely on the static signature analysis of the running programs, requiring continuous software updates which is not affordable for embedded systems and edge devices with limited computing and communication bandwidth. To address these challenges, this chapter proposes an accurate and cost-efficient machine learning-enabled countermeasure for securing modern edge devices against emerging cyber-attacks, i.e., malware and Side-Channel Attacks (SCAs) at the hardware level by monitoring applications' Hardware Performance Counter (HPC) features. We explore the effectiveness of attack detection on edge devices, including autonomous vehicles, mobiles, and desktops.

4.1 Evaluation of Machine Learning-based Detection against Side-Channel Attacks on Autonomous Vehicle

Autonomous vehicles have achieved great success in academia and industry due to the progress made in cheaper sensors, higher computation capability of processors, and effective object

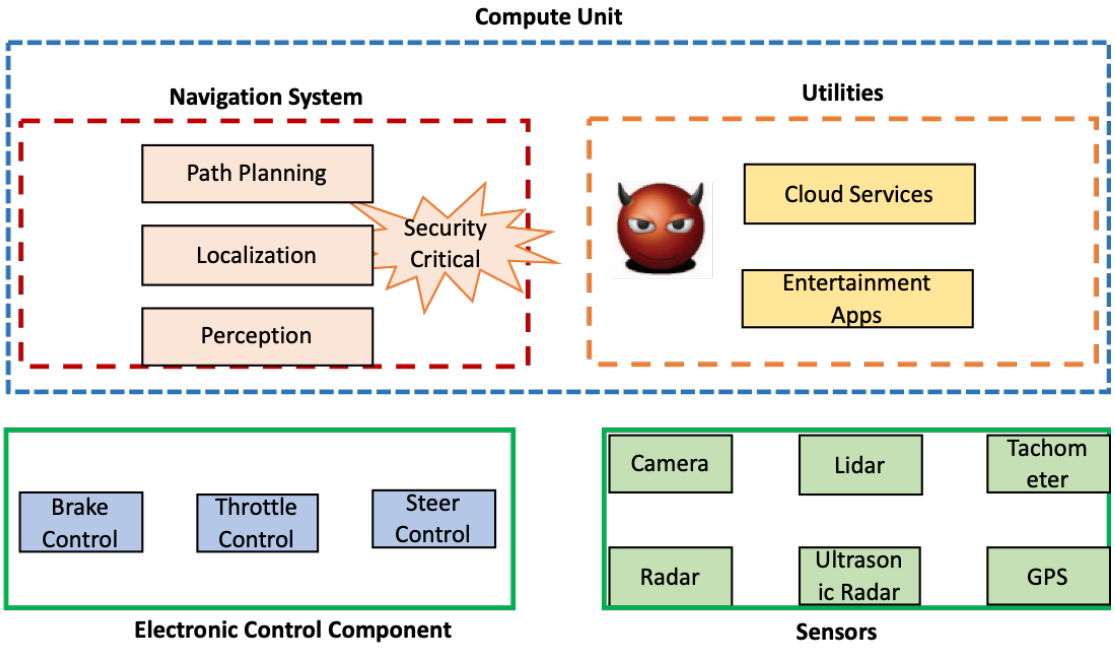


Figure 4.1: General hardware and software architecture of an autonomous vehicle

detection. Many passenger vehicles are also equipped with autonomous driving capabilities, like Tesla [97], Uber [98], Waymo [99], Baidu [100], etc. As shown in Figure 4.1, there are three main components in autonomous vehicles, including sensors, electronic control component, and compute unit. Sensors, including cameras, radar, GPS, etc., are used for collecting information from the real-world environment. The control component is in charge of operations on vehicles, like putting the brake on a car. For the computing unit, there are two parts: navigation system and utility. The navigation system is equipped with path planning, localization, and perception ability based on the information collected from sensors and reinforcement learning and deep learning techniques. The computation result from the navigation system is a command like turning right, stopping, etc., which are sent to the electronic control component. Since it can directly influence the operations of a car, they are security-critical and provided by vendors. The other part running on the computing unit is utility including cloud services and entertainment apps, which a third party could provide. Hence, they share computation units with the navigation system.

Though great success obtained, recent advancements have shown that the autonomous driving architecture introduces new vulnerabilities and attack surfaces for cache-based side-channel attacks (SCAs). Recent work [10] demonstrates that using Prime+Probe [1,21] can infer the location and route that the user is taking by inferring the data access pattern of the Adaptive Monte-Carlo

localization (AMCL) algorithm. It could infer the route/location of drivers with up to 81% accuracy for route prediction and 75% accuracy for location prediction. In recent years, there are a number of new types of SCAs developed, like Flush+Reload [2], Flush+Flush [3], Prime+Probe [1], etc. that attempt to infer users' sensitive information.

4.1.1 Machine Learning based Detector

This section introduces the hardware architectures used in commercial autonomous vehicles and then presents the details of our proposed ML-based detector with runtime microarchitectural behaviors against SCAs.

Threat Model

This work mainly focuses on the software attacks that exploit hardware vulnerability (shared memory and cache) and infer location, route, and other private information without permission to sensor data and access to the physical measurement. As introduced in Section 4.1, there are two types of applications residing in compute unit: navigation system and utilities. Since the navigation system is developed and maintained by vehicle vendors, applications from the utility are more likely to be inserted with SCAs' codes. Hence, this work considers SCAs installed in the utility part and residing in the same compute unit with the navigation system, indicating the shared cache hierarchy between attacks and programs in the navigation system.

Table 4.1: Common Compute Unit for Autonomous Vehicles

Vendors	Architecture	Compute Unit
Baidu [100]	ARM	Arm Cortex-A53, FPGA, GPU
Telsa [97]	ARM	Cortex-A72 CPU, GPU, Accelerator
Waymo [99]	x86	Intel Xeon CPU, GPU
Uber [98]	x86	Intel CPU, GPU

Hardware Performance Counters

Modern microprocessors are equipped with a set of special-purpose registers for measuring hardware-related events, i.e. hardware performance counters (HPCs). Both the ARM and Intel x86

architecture provide the performance monitoring unit (PMU) interface to access HPCs. Since the Pentium, Intel processors enable PMU feature while the ARMv6 is the first PMU-enabled ARM processor and the afterward ARM11, Cortex-R, and Cortex-A cores also provide the functionality. As listed in Table 4.1, we report the most prevalent processors used by top car vendors and research. It shows that ARM and x86 Intel architectures are the two most popular ones, and GPU, FPGA are included for the acceleration of complex computation. Intel processors are also commonly used for research [101, 102]. It is also noticeable that all processors listed in the Table 4.1 have access to hardware performance counters, indicating that using hardware performance counters to detect side-channel attacks is a viable approach.

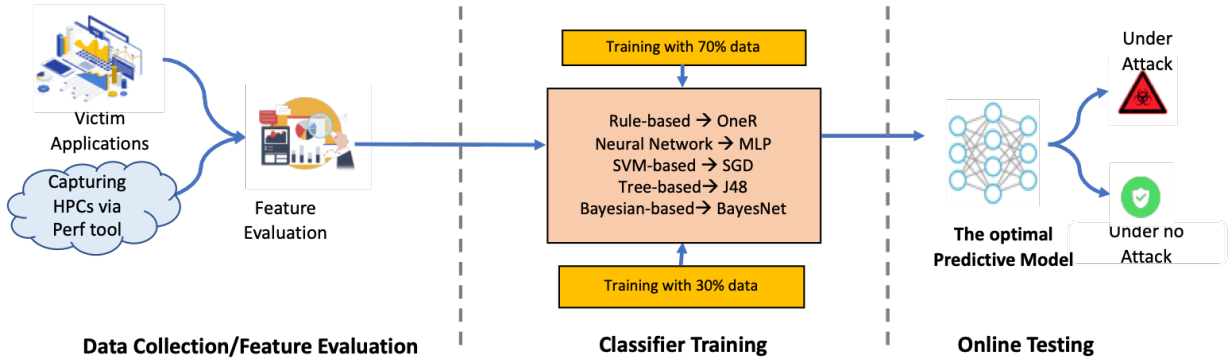


Figure 4.2: Overview of the ML-based Detector

4.1.2 Building ML-based Detector

As shown in Figure 4.2, there are three main steps to constructing a ML-based SCAs detector: a) data collection and feature evaluation; b) training classifier; c) online testing for the optimal predictive model.

Data Collection and Feature Evaluation

The tested attacks are Flush+Reload, Flush+Flush, and Prime+Probe while victim applications use Fast R-CNN [103] for Perception , the adaptive Monte-Carlo Localization (AMCL) [104] for localization, optimization method [105] for path planning . Since their computation patterns might reveal users' environment, location, or other privacy as demonstrated in prior work [10]. Run-time microarchitectural behaviors of victim applications are collected by Perf [106] tool to form a database with a known label ("under no attack" or "under attack"). Based on the behavior and

Table 4.2: List of HPC events collected for SCAs detection

L1 HIT	L1 MISSES
L2 HIT	L2 MISSES
L3 HIT	L3 MISSES
All BRANCHES RETIRED	BRANCHES MISPREDICTED
BR_NONTAKEN_CONDITIONAL	BR_TAKEN_CONDITIONAL
TAKEN_INDIRECT_NEAR_CALL	UOPS_RETIRED.ALL
INST_RETIRED.ANY	DTLB_LOAD_MISSES
DTLB_STORE_MISSES	ITLB_MISSES

functionality of studied SCAs, 16 HPC features are considered in this work for further analysis as listed in Table 4.2. Since the HPCs can be collected simultaneously, it is essential to identify the most prominent HPCs. These hardware performance counters data are collected using the four available HPC registers in the experimented Xeon processor at every sampling interval (10 ms). Next, both "under attack" and "under no attack" HPC data from each same sampling intervals are merged to create the final dataset for the corresponding sampling interval.

Training Classifier Selection

The ML classifiers evaluated in this work are selected from five different categories, including NaiveBayes, Multi-Layer Perceptron (MLP), SGD, OneR, and J48. The rationale for choosing these machine learning models is that they are from different branches of ML, including Bayesian network-based, neural network, support vector machine, rule-based, and tree-based techniques covering a diverse range of learning algorithms that are inclusive of modeling both linear and nonlinear problems. The prediction model produced by these learning algorithms can be a binary classification model that is compatible with the SCAs detection problem in our work. Furthermore, Weka data mining tool is leveraged for implementing the ML classifiers. A standard 70%-30% dataset split for training and testing is conducted to validate each of the utilized ML classifiers. Next, 70% of the randomized data is used for training the classifiers for the percentage split testing, and the rest of 30% is used for testing evaluation. In addition, a k-fold (k=10) cross-validation is also conducted on the training dataset.

Online Testing

Once classifiers are trained with the 70% dataset and tested with the rest 30%, the optimal classifier is chosen as the predictive model, which will be deployed online against SCAs based on accuracy, robustness, and computation latency.

4.1.3 Results Evaluation

In this part, all experiments are conducted on an Intel E5-2650 with 8 cores, and 16GB DRAM. To demonstrate the effectiveness of ML-based detectors against SCAs on autonomous computing platforms, we build our ML-based detectors with microarchitectural events captured with Perf tool [106] and evaluate their detection accuracy, robustness, and computation efficiency.

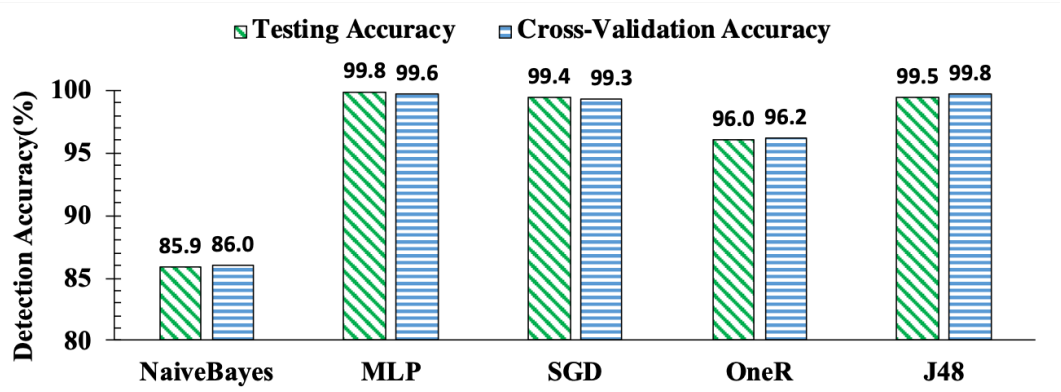


Figure 4.3: Testing accuracy and cross-validation accuracy

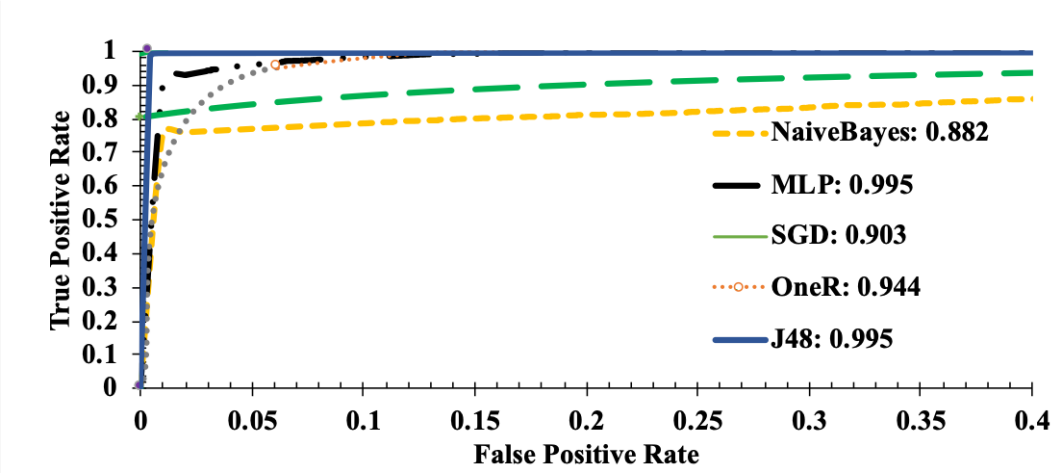


Figure 4.4: ROC Curve and AUC value of various classifiers

Detection Results

-*Detection Accuracy* As shown in Figure 4.3, the percentage split testing and cross-validation accuracy of the five implemented ML classifiers are presented. It is observed that testing accuracy and cross-validation accuracy have shown similar trends across all five classifiers. Moreover, NaiveBayes achieves lowest accuracy with less than 90% for both testing and cross-validation accuracy. The three classification classifiers, namely MLP, SGD, and J48, achieve high detection accuracy (>99%), while OneR has around 3% less accuracy compared to the three classifiers. - *Robustness* Receiver Operating Characteristics (ROC) Curve is produced by plotting the fraction of true positives rate versus the fraction of false positives for a binary classifier. The best possible classifier would yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives. Area under the ROC Curve (AUC) metric. The AUC corresponds to the probability of correctly identifying "under attack" and "under no attack" and robustness is referred to how well the classifier distinguishes between the two classes for all possible threshold values. Higher AUC indicates better robustness for ML classifiers. Figure 4.4 depicts the ROC Curve of various ML classifiers with corresponding AUC values. It can be observed in Figure 4.4 that the NaiveBayes algorithm performs the worst in terms of ROC Curve, having the most significant distance to the point (0,1). The J48 classifier's AUC value is closer to the coordinate (0,1), indicating a higher true positive rate and less false positive rate than the other four classifiers evaluated in this work. The ROC curve and AUC value of MLP are similar to those in J48, indicating that the mispredicted instances are evenly distributed between "under attack" and "under no attack" classes. - *Efficiency* Lastly, to accordingly account for both performance rate and cost of ML classifiers,

Table 4.3: F-measure of various classifiers

Classifiers	NavieBayes	MLP	SGD	OneR	J48
F-measure	0.862	0.934	0.894	0.945	0.993

in Figure 4.5 we compare detection rate over a computational latency (F-measure/Latency) for various ML classifiers. F-measure is interpreted as a weighted average of the precision (p) and recall (r) which is formulated as $\frac{2\times(p\times r)}{p+r}$. F-measure is a more comprehensive evaluation metric over accuracy (percentage of correctly classified samples) since it takes both the precision and

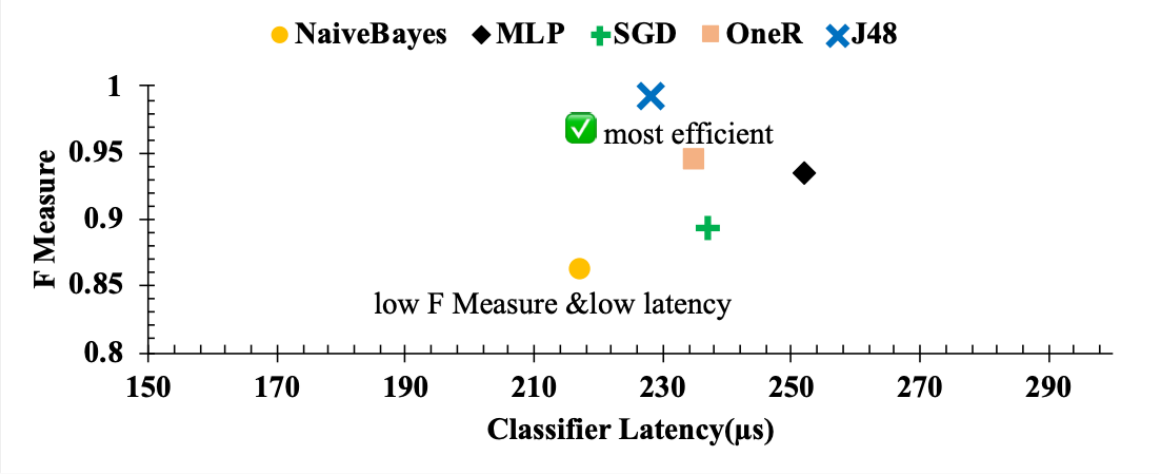


Figure 4.5: Efficiency comparison among various classifiers

the recall into consideration. The F-measure of investigated classifiers are presented in Table 4.10. We use F-measure over latency to identify the SCA detectors that require small cost can detect the program's maliciousness with high accuracy and performance. A higher ratio classifier is considered more efficient than a classifier with a lower ratio. As shown in Figure 4.5, a clear trade-off is seen between F measure and latency achievable for real-time hardware-assisted SCAs detection. The NaiveBayes show the smallest timing costs with the lowest SCAs F-measure. For highly resource-constrained embedded systems, techniques such as J48 provide low computational overhead (only 10 *us* higher than NaiveBayes), while achieving an F-measure of close to 0.993 on average.

4.2 Proposed Micro AI-based Countermeasure against Malware and Side-Channel Attacks

Recent advancements in digital electronics have enabled a widespread proliferation of embedded systems ranging from micro-sensors, cell phones, and PDAs to smart homes, healthcare, and military applications. The constant interaction between physical and cyber worlds has made security one of the major concerns in the design of embedded systems. For the past decades, cybersecurity has been on the front line of global attention as a critical threat to the information technology infrastructures [111,112]. Attackers are increasingly motivated and enabled to compromise software and computing hardware infrastructure. Recent studies have shown that the attackers

Table 4.4: Comparison of recent hardware-assisted malware and side-channel attack detection techniques and their implementation methods

Research	**Processor**	**Platform**	**Classification Model**	**Malware**	**SCAs**	**Feature Evaluation**	**Overhead Analysis (SW/HW)**	**Evaluation Metric**	**ACC**	**FP Rate**
[107]	Intel	Linux	ocSVM	✓	✗	Fisher Score	–	ACC, F Score, AUC, ROC	99.5%	–
[108]	Intel	Windows	LR, NN	✓	✗	Expert Knowledge	Hardware	ACC, FP, ROC	94%	7%
[109]	Intel	Windows	LR, NN, EL	✓	✗	Expert Knowledge	Hardware	ACC, FP, ROC, AUC	86.7 %	–
[110]	Intel	Linux	SVM, ocSVM, NB, DT	✓	✗	Gain Ratio	Hardware	Confusion Matrix, ROC	99%	0.52%
[73]	Intel	Linux	BN, J48, JRip, MLP, OneR, RT, SGD, SMO, AB, BG	✓	✗	Correlation Attribute Evaluation	Hardware	ACC, AUC, ROC, ACC*AUC, Area	88%	–
[9]	Intel	Linux	DTW	✗	✓	Fisher Score	–	ACC, F Score, ROC	100%	20%
[17]	Intel	Linux	LDA, LR, and SVM	✗	✓	Expert Knowledge	–	ACC, FP, ROC	99.51%	0.40%
[19]	Intel	Linux	CPD	✗	✓	Relief Algorithm	–	ACC, FP	100%	4.5%
[41]	Intel	Linux	NN	✗	✓	–	–	ACC, F Score, FP	99%	–
This Work	**Intel, ARM**	**Linux, Android**	**NaiveBayes, MLP, J48, OneR, JRiP, SGD**	✓	✓	**Correlation Attribute Evaluation**	**Software and Hardware**	**ACC, AUC, ROC, Latency, FP, Power, Area**	**92.2%, 98.9%**	**4.6%, 0.8%**

SW: software, HW: hardware, Accuracy: ACC, Sensitivity: S, Specificity: C, K Nearest Neighbor: KNN, BayesNet: BN, NaiveBayes: NB, Logistic Regression: LR, AdaBoost: AB, Bagging: BG, Support Vector Machine: SVM, One Class SVM: ocSVM, Neural Netework: NN, Last Level Cache References: LLC, REPTree: RT, Decision Tree: DT, Random Forest: RF, Ensemble Learning: EL, Dynamic Time Warping: DTW, Linear Discriminant Analysis: LDA, Change Point Detection theory: CPD.

take advantage of emerging hardware vulnerabilities to compromise systems and deploy malicious activities [1–3]. Hence, the security of a computer system can be compromised at the hardware level through various types of attacks, such as by executing malicious applications to infect the target host or deploying microarchitectural side-channel attacks (SCAs) [1–3, 113, 114] to infer confidential information.

Malware refers to any piece of software written to steal data, unauthorized data access, damage devices, etc. Viruses, Trojans, Spyware, Rootkits and Ransomware are among the different types of malware [115]. Microarchitectural SCAs have also posed serious threats to the security of modern computing systems. Such attacks exploit side-channel vulnerabilities originating from fundamental performance-enhancing components such as cache memories. Many of the existing cyber-attacks on conventional computing platforms such as servers and desktops can be launched on the embedded systems due to their similarities in using general-purpose processors computing systems and their connectivity to the internet.

The significant growth of modern computing systems in embedded applications and IoT domains has further highlighted the severe impact of cybersecurity threats [73, 116–118]. Depending on the target application, these devices can produce a massive amount of data that needs to be handled, and the emerging edge computing paradigm is receiving a tremendous amount of interest to tackle this challenge. Edge devices could include many different computing platforms, such as IoT sensors, laptops, smartphones, security cameras, etc. Ensuring security in embedded systems and edge computing devices translates into several design challenges imposed by the unique features of these systems. There exist some important factors influencing the security vulnerability of embedded systems and IoTs, including the limited energy and resources available, the low computational capacity, and a significant number of computing nodes in the network. Hence, to keep on combating the increase in malicious cyber-attacks, effective intelligent security countermeasures need to be proposed to protect the integrity and confidentiality of the authenticated users' information [118,119].

Traditional software-based detection techniques have shown to be inefficient and imposed significant complexity and computational overheads on the system. For instance, signature-based detection and semantics-based solutions (e.g., off-the-shelf antivirus tools) are ineffective for resource-constrained embedded systems due to the limited available computing resources. In addition, such detection methods depend on the static signature analysis of executed applications that make them

incapable of detecting complex unknown attacks. Recent advancements in microarchitectural security have demonstrated that malicious activities at the processor hardware level ranging from application-based malware to microarchitecture-level side-channel attacks can be accurately identified with the aid of standard Machine Learning (ML) algorithms [9, 16, 73, 109, 117]. In particular, Artificial Intelligence (AI) and ML, driven by a significant increase in the size of data from high-performance computing systems, have been widely adopted in various application domains, with hardware-assisted security being no exception.

ML-based security countermeasures at the hardware level apply the standard ML techniques on the low-level features such as microarchitectural events collected by Hardware Performance Counter (HPC) registers. HPCs are a set of special-purpose registers in the processing units to capture the trace of hardware events for a running application [120]. HPCs registers have primarily been designed to conduct architectural performance analysis and tuning. Recent works have proposed utilizing the HPCs information to secure the hardware systems against malware software and microarchitectural SCAs. In addition, the latency of detecting malicious code is negligible by order of magnitude with relatively low hardware costs [121].

Nonetheless, when it comes to detecting malicious patterns at the hardware level in edge devices, the available underlying resources become a more critical point of concern. Compared to high-performance servers, edge devices host much fewer computation resources for processing heavy computations and complex workloads. In this paper, we have identified and addressed major challenges of ML applications for accurate and cost-efficient malware and side-channel attacks detection in edge devices that have been ignored in prior studies. In particular, limited computing power and resources in embedded systems and edge devices, as well as the small number of available hardware performance counter registers on the modern microprocessors chip (only 2-8 HPCs) that can be simultaneously monitored, have made accurate and cost-efficient runtime malware and side-channel attack detection in edge devices a challenging problem. Moreover, while a wide range of classification and anomaly detection techniques are developed by applying ML techniques, existing works in particular on malware and/or SCA detection have primarily focused on one or a few ML techniques for attacks detection and classification [9, 16]. Such an analysis leaves a void in terms of performance and overhead (latency, hardware implementation, power, etc.) of ML-based security countermeasures, as various ML classifiers yield different performance vs. overhead trade-offs

in detecting various types of attacks. Compared to high-performance servers, edge devices host fewer computation resources for processing heavy computations and complex workloads. Therefore, adopting complex learning classifiers (e.g., neural network) presented in prior works [9] could further result in congestion and racing up for CPU resources between malware/SCAs detector and other running applications. As a result, there is an urgent need for developing an accurate and cost-efficient micro AI-enabled countermeasure to protect modern edge devices against emerging cyber-attacks.

In response to the discussed challenges, this work performs a comprehensive assessment of various machine learning-based countermeasures for accurate and cost-efficient malware and side-channel attacks detection using microarchitectural features and propose a lightweight hardware-assisted micro AI-enabled countermeasure against emerging malware and side-channels attacks for securing modern edge devices. The results of this research could help designers better realize and navigate the trade-offs between several design parameters offered by each machine learning algorithm to develop effective ML-based countermeasure against emerging cyber-attacks such as malware and side-channel attacks for modern edge devices, especially in resource-constrained systems. From Table 4.4, we can conclude that our proposed work conducts an effective feature evaluation to analyze and select the most prominent HPC features for malware and microarchitectural side-channel attack detection. Moreover, it further performs a comprehensive analysis of various ML models for enabling an efficient and accurate micro AI-based solution to enhance the security of edge devices against emerging attacks. Additionally, our work implements the ML-based detectors in both software level and hardware level to present the suitability of our proposed micro AI for edge devices in terms of accuracy, latency, power, and area overheads. The main contributions of our work are summarized as follows:

- We profiled a large number of applications and built an extensive database of HPC samples collected from benign, android malware, and microarchitectural side-channel attacks executed. For this purpose, two widely used edge computing processors are used, including Google Pixel 4 with ARM processors for malware and Intel Core-I5 for side-channel attacks experiments.

- To eliminate the impact of limited HPCs in modern edge processors, HPC features is evaluated using an effective feature selection method. The essential HPCs features are identified for effective malware and SCAs detection.

- We further explore the HPCs monitoring overhead when microarchitectural features are sampled at different intervals to determine the appropriate sampling interval for malware and SCAs detection.

- For a thorough analysis, various types of ML classifiers are implemented and precisely compared across different evaluation metrics including detection accuracy, F-measure, ROC curve analysis, computational latency, and hardware overhead to determine the most accurate and cost-efficient ML classifiers to enable on-device micro AI countermeasures for detecting signature of emerging cyber-attacks including malware and SCAs and securing modern edge devices at hardware level.

4.2.1 Background

Application of Proposed Micro AI Solution on Edge Devices

Figure 4.6 illustrates the applicability of the proposed micro AI countermeasure for securing edge devices at the hardware level against emerging malware and side-channel attacks. The proposed ML-based solution secures the system by estimating the type of running application (benign vs. attack) at runtime using limited HPC features collected at the hardware layer of edge processors. As shown in Figure 4.6, the network of embedded systems and IoT consists of a variety of edge devices, including smartphones, laptops, wearable devices, etc., that are connected via wired or wireless network [122]. Early edge devices are only in charge of collecting and sending data to a remote server for further analysis. With the development of hardware design and manufacturing technology, recent edge devices are also equipped with computation and analysis capability, enabling the computation near end users instead of sending data and workloads to central servers, which reduces network latency and response time.

Though benefits are brought by deploying more advanced edge devices, new security concerns are raised since more sensitive data and critical applications reside in these devices. Malware or SCAs can obfuscate user applications and unconsciously execute users with benign applications. We design a micro AI detector that takes HPCs from processors as input features and sends them to ML classifiers to decide whether malware or SCAs is in the system to address the challenge. Prior detection approaches used in traditional computation platforms, like taint analysis [123, 124] and deep learning-based detection [125], are not applicable for edge devices due

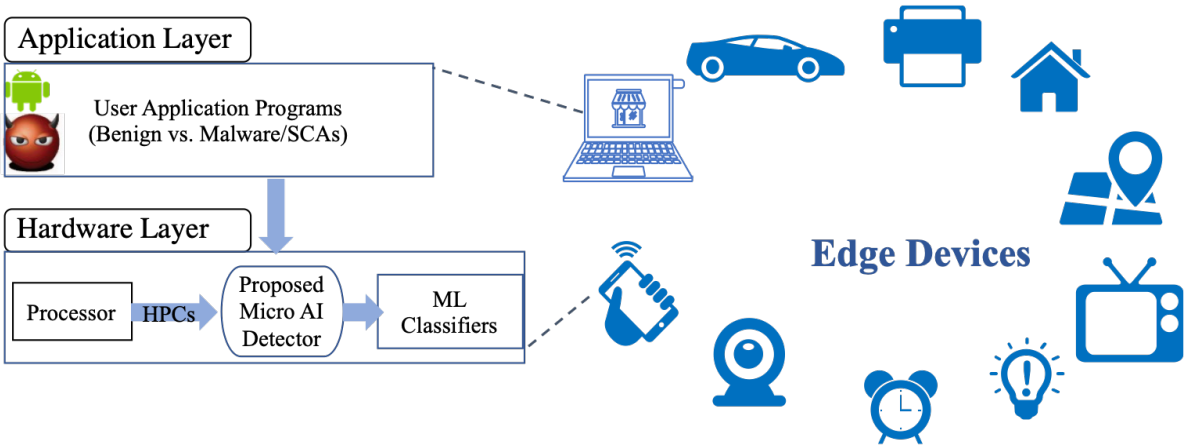
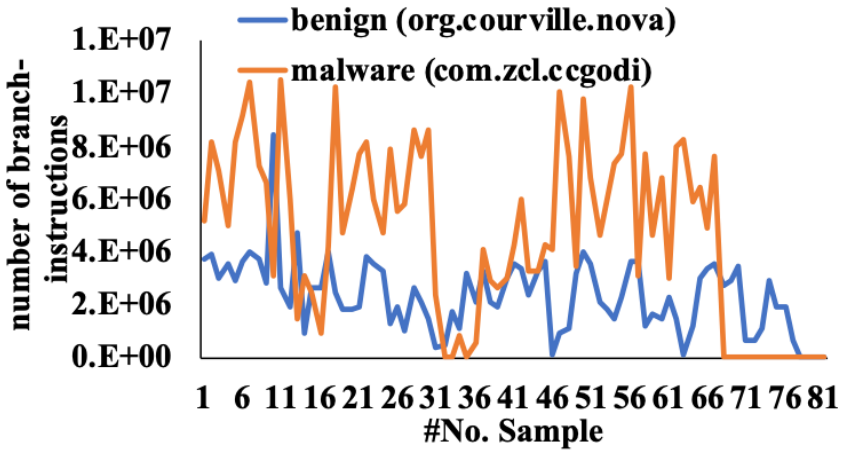


Figure 4.6: Application of the proposed micro AI enabled countermeasure for securing edge devices at the hardware level.

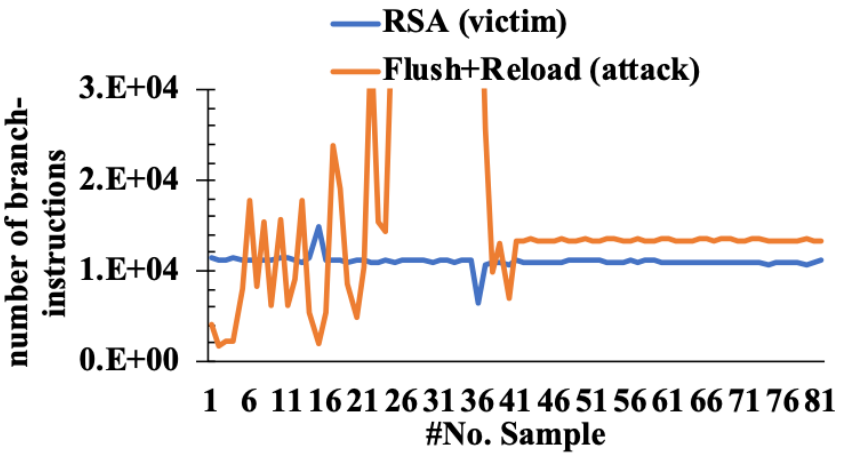
to their high computation resource demand. Compared to traditional computation platforms, like desktops and servers, such edge devices are usually battery based or have limited energy sources due to the portability and low cost design consideration. This motivates the application of accurate and cost-efficient ML-based micro AI countermeasures for securing edge devices at the hardware level, which provides higher efficiency and less visibility to the potential exploit.

Hardware Performance Counter Registers

Hardware Performance Counters (HPCs) are a set of special-purpose registers to record the occurrences of hardware events, including instructions, branches executed, and cache misses. They have been extensively used to predict the power estimation [126], performance tuning [127], debugging [128], and energy efficiency of computing systems [129]. They also help to enhance the systems' security by providing microarchitectural information of malware, side-channel attacks, and building detectors based on the events' information [32, 109, 130]. Both ARM and Intel x86 architecture provide the performance monitoring unit (PMU) interface to access HPCs. Since the Pentium Intel processors enable the PMU feature, the ARMv6 is the first PMU-enabled ARM processor, and the afterward ARM11, Cortex-R, and Cortex-A cores also provide the functionality [120, 131]. Linux-based Perf [132], PAPI [133], and Intel VTune [134] are provided as tools to collect HPCs PCs or severs while Simpleperf [135] developed by Google Android team is for Android platform.



a) Benign and malware



b) RSA (victim) and Flush+Reload (attack)

Figure 4.7: Branch-instruction HPC traces between benign(victim) and attacks

We present the number of $branch - instruction$ for each sampling data of benign and malware, and victim and SCAs in Figure 4.7 where sampling rate is 1000 per second, to further demonstrate the potential of leveraging hardware performance counters to enable HPCs-based micro AI for securing edge devices. Figure 4.7-(a) shows branch-instruction traces of malware and benign, indicating that the malware has higher branch instruction than the benign application with totally a different and distinguishable trend. Similar observation can be found in Figure 4.7-(b) where Flush+Reload SCA shows higher branch-instruction events than RSA victim application. Both Figures 4.7-(a) and (b) demonstrate that there exists a clear difference between the microarchitectural behavior of benign and malware, and RSA (victim) and Flush+Reload (SCAs). This observation highlights the suitability of using HPCs data to distinguish the trace of malicious software from

benign programs by applying effective ML algorithms.

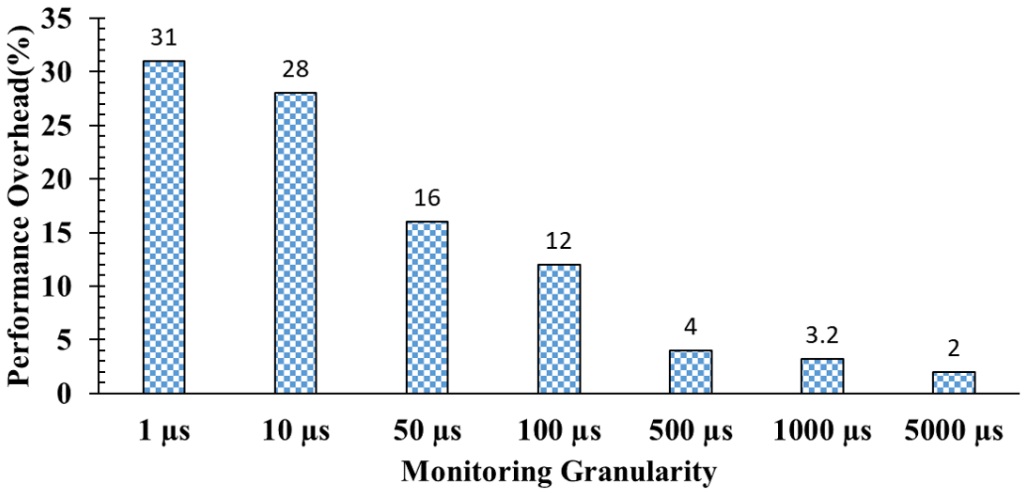


Figure 4.8: Performance overhead with various monitoring granularity

HPCs Monitoring Overhead

For minimizing the overhead incurred by HPCs monitoring, we investigate the relation between sampling rate and overhead. As shown in Figure 4.8, the x-axis represents applied monitoring granularity ranging from 1 μs to 5000 μs, the primary y-axis represents the execution time of victim applications, and the second y-axis represents performance overhead under different monitoring granularities. Execution time under no HPCs monitoring is used to obtain performance overhead percentage. It is observed that, generally, the smaller the monitoring granularity, the larger the performance overhead. When the monitoring scale is 1 μs, performance overhead is at its highest value reaching 30%. Due to the significant difference in monitoring overheads, it is vital to determine a proper level of monitoring granularity to balance the detection performance and HPCs monitoring costs. This work chooses one millisecond as the sampling interval to ensure overhead is around 3%.

4.2.2 Proposed Methodology

In this section, we first present the ML-based countermeasures for protecting edge devices against malware and SCAs threats as shown in Figure 4.9. As established, the proposed approach is comprised of two main parts: off-line training and online deployment. For the training part: benign applications, malware, and SCAs are profiled by collecting HPC data every one millisecond. And

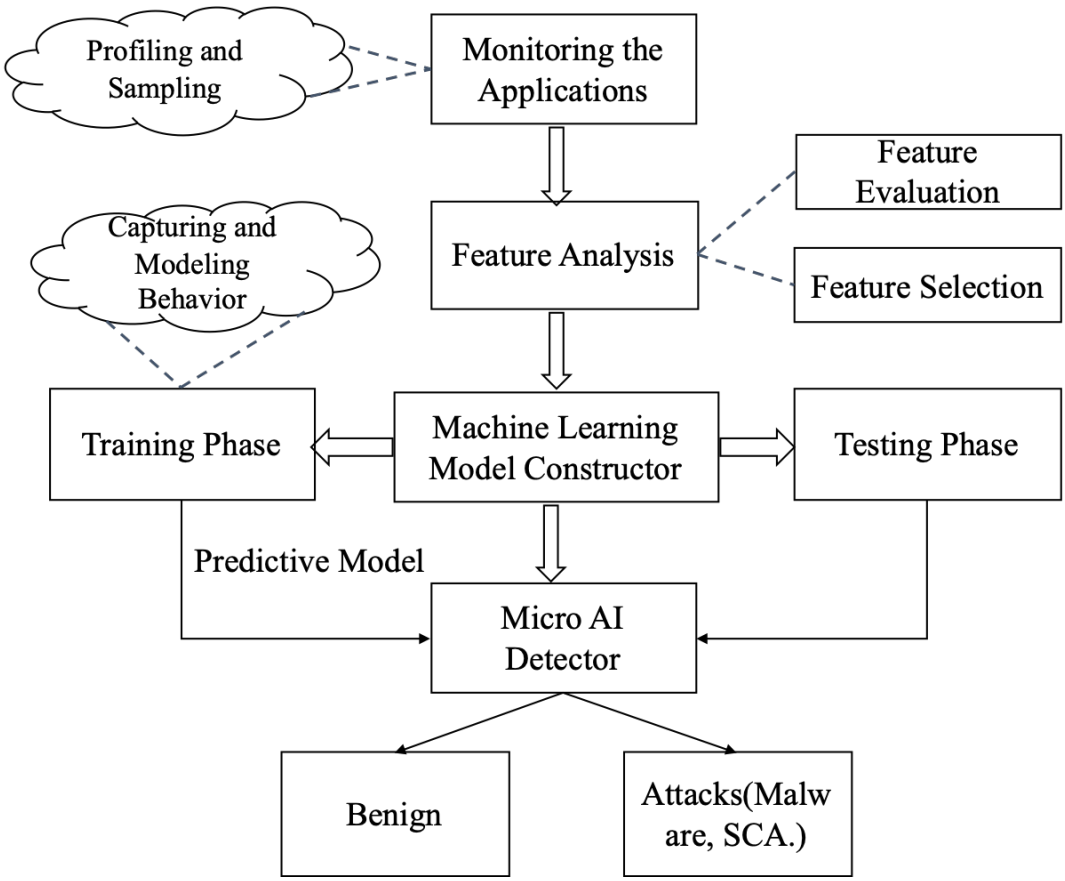


Figure 4.9: Overview of the proposed machine learning-based countermeasure for edge devices

then, the HPCs data is used to train ML classifiers for building ML-based attacks detection models. Various classifiers are explored to find the optimal one with high accuracy and low computation cost. The selected model will be deployed on edge devices for capturing attacks at runtime.

Table 4.5: Experiment setup

Hardware	Processor	Operating system	Threats
Pixel 4	Qualcomm Snapdragon	Android 11.0	Malware
Desktop	4 cores	Ubuntu 16.0	SCAs

Data Collection

In this work, we leverage two platforms as examples of edge devices as shown in Table 4.5. As introduced in Section 4.8, one millisecond is chosen as the sampling interval and 16 HPC features are considered in this work for further analysis as listed in Table 4.6. We execute each application

Table 4.6: The collected HPC features

HPC event	Description
branch-instructions	# branch instructions retired
branch-loads	# successful branches
branch-misses	# branches mispredicted
instructions	# instructions retired
bus-cycles	time to make a read/write between the cpu and memory
cache misses	# last level cache misses
cache-references	# last level cache references
l1-dcache-load-misses	# cache lines brought into L1 data cache
l1-dcache-loads	# retired memory load operations
l1-dcache-stores	# cache lines into L1 cache from DRAM
l1-icache-load-misses	# instruction misses in L1 instructions cache
llc store misses	# misses occurred in L3 cache during store operation
llc-load-misses	# cache lines brought into L3 cache from DRAM
llc-loads	# successful memory load operations in L3
iTLB loads	# load operations occurred in instruction TLB
iTLB-load-misses	# misses in instruction TLB during load operations

(benign or malware) four times, and only 2 and 4 HPC events are selected to collect during each run from the tested ARM and Intel processors, respectively. In this way, we can monitor 16 HPC events after eight or four rounds of applications' execution. Each database introduced below is split into 70%-30% two parts where the 70% part of benign and malware/SCAs is used for training and the rest 30% is used for testing. The databases for Android and Linux are introduced in detail. We leverage Perf [132] and Simperperf [135] to collect HPCs for Linux and Android, respectively. Both of the tools can collect multiple events simultaneously, and the number of events to be collected concurrently depends on the design of architectures and the availability of HPC registers. For example, ARM Cortex-A5 processors provide two while ARM Cortex-A7 processors provide 4 HPC registers physically available on the processor chip to collect the hardware related events. We leverage these performance analysis tools to profile an extensive set of malicious (malware and

side-channel attacks) and benign applications.

- *Android Database* As for the malware, we selected 100 Android malware randomly from VirusTotal [136] and 100 Linux malware. We downloaded the top 200 most popular free apps in Google Play by the end of Feb 2021 as our target programs to form benign applications. Then, we employ the apps with VirusTotal [136] to confirm if the apps are benign. Lastly, the top 100 apps which are confirmed as benign are selected as the benign apps dataset.

- *Linux Database* MiBench [82] benchmark suite is used to represent benign applications with 200 benign applications are selected. For SCAs, we use attacks from Mastik [74] including Flush+Reload, Flush+Flush, L1 Prime+Probe, and L3 Prime+Probe. And each type of SCAs is monitored 50 times to balance the benign and SCAs samples.

Feature Selection

Detecting malware and side-channel attacks at the hardware level using ML models requires representing programs at low-level features, leading to high-dimensional data processing and increasing computational overheads and complexity. Furthermore, incorporating irrelevant features would lead to lower accuracy and performance for the classifiers. Hence, it is crucial to perform an effective feature reduction of collected data to alleviate unnecessary computational overheads and determine the most prominent low-level features [18, 137]. To detect the attacks in real-time with minimal overhead, we intend to identify a minimal set of critical HPCs that are feasible to collect even on low-end processors with a small number of HPCs in a single run. Therefore, a subset of HPC features is selected, representing the most important features for classification. The selected features are then supplied to each ML-based attacks detector. The detector attempts to find a correlation between the feature values and the application behavior to predict the attacks.

Given the limited number of HPCs available in modern microprocessors (e.g., only 4 HPCs on tested Intel I5-3470) to be collected at one time simultaneously, it is necessary to identify the most important ones among all available HPCs listed in Table 4.6 for classifying the attacks and benign applications conditions for different types of attacks [73]. For HPCs reduction, we employ Correlation Attribute Evaluation (*CorrelationAttributeEval* in Weka) with its default settings to calculate the Pearson correlation between attributes (HPC features) and class (attacks and benign applications conditions). Correlation attribute evaluation algorithm calculates the Pearson

correlation coefficient between each attribute and class, as given below:

$$\rho(i) = \frac{cov(Z_i, C)}{\sqrt{var(Z_i)\ var(C)}} \quad i = 1, ..., 16 \tag{4.1}$$

where ρ is the Pearson correlation coefficient. Z_i is the input dataset of event i ($i = 1, \ldots, 16$). C is the output dataset containing labels, i.e. "malware/SCAs" or "benign/victmi" in our case. The $cov(Z_i, C)$ measures the covariance between input data and output data. The $var(Z_i)$ and $var(C)$ measure variance of both input and output datasets, respectively. After the evalaution and reduction, the top eight HPCs for malware and SCAs detection are listed in Table 4.7.

Table 4.7: The collected HPC features and their ranking

Ranking_Malware	**HPC Name**	**Ranking_SCAs**	**HPC Name**
1	L1-dcache-stores	1	L1-dcache-loads
2	branch-instructions	2	instruction
3	L1-dcache-loads	3	branch-instructions
4	iTLB-load	4	branch-misses
5	iTLB-load-misses	5	cache misses
6	L1-icache-load-misses	6	L1-dcache-store-misses
7	L1-dcache-store-misses	7	L1-dcache-load-misses
8	branch-misses	8	iTLB-load-misses

Training and Testing ML-based Micro AI Classifiers

Table 4.8 describes the ML classifiers evaluated in this work that is selected from seven different categories. These ML classifiers include NaiveBayes, Logistic, MultiLayerPerceptron (MLP), SGD, JRip, OneR, J48. The rationale for selecting these machine learning models is that they are from different branches of ML, including Bayesian network-based, neural network, support vector machine, lazy learning-based, rule-based, and tree-based techniques covering a diverse range of learning algorithms which are inclusive of modeling both linear and nonlinear problems. In addition, the prediction model produced by these learning algorithms can be a binary classification model which is compatible with the attacks detection problem in our work. Furthermore, the Weka data

Table 4.8: Evaluated ML classifiers for attacks detection

ML Category	Notation	Selected Classifier
Bayesian Network	Algorithms that use Bayes Theorem in some core way, like Naive Bayes.	NaiveBayes
Neural Network	Series of algorithms that attempt to recognize and mimic the human brain operations.	Multi-layer Perceptron (MLP)
Support Vector Machine	Linear model for classification and regression problems.	SGD
Rules	Algorithms that use rules, like One Rule.	JRiP, OneR
Trees	Algorithms that use decision trees, like Random Forest.	J48

mining tool is deployed for implementing the ML classifiers. A standard 70%-30% dataset split for training and testing is followed to validate each of the utilized ML classifiers. Next, 70% of the randomized data is used for training the classifiers for the percentage split testing, and the rest of 30% is used for testing evaluation.

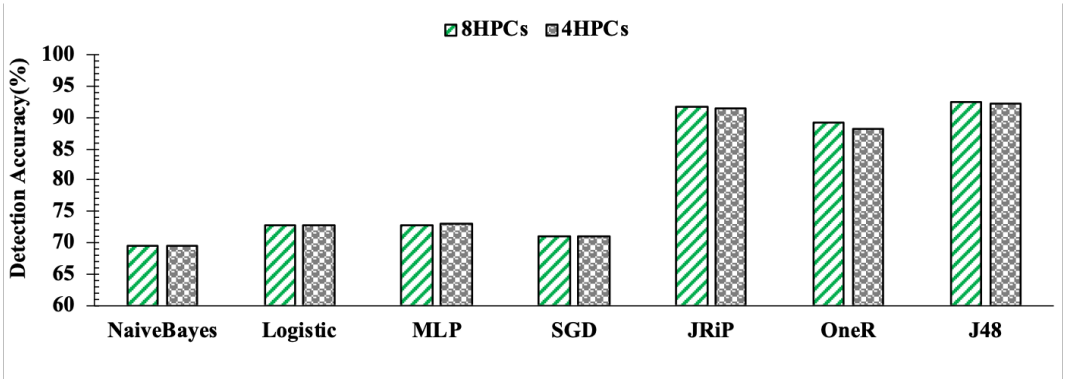


Figure 4.10: **Malware** detection accuracy of different classifiers with various HPCs

4.2.3 Experimental Results and Evaluation

To comprehensively analyze the effectiveness of ML-based micro AI solutions for securing edge devices against emerging cyber-attacks, in this section, we present the evaluation and comparison results across different evaluation metrics, including detection accuracy, false positive rate, F-measure (F-score), robustness and efficiency, hardware overhead, and cost.

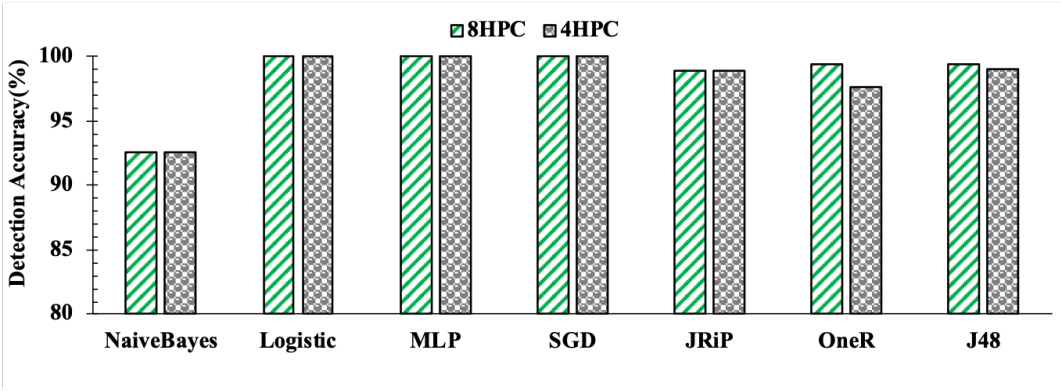


Figure 4.11: **SCAs** detection accuracy of different classifiers with various HPCs

- *Detection Accuracy* Detection accuracy in ML-based security countermeasures is defined as the percentage of correctly classified samples. To select the most effective classification model and evaluate the influence of a different number of HPCs used (8 vs. 4 features), in Figure 4.10 and Figure 4.11, we demonstrate the observed detection accuracy against malware and SCAs respectively across seven implemented ML classifiers trained and tested with 8 and 4 HPCs. In particular, we focus on analyzing the effectiveness of proposed ML-based micro AI countermeasures by considering 8 and 4 HPC features, since most modern microprocessors deployed in edge platforms are equipped with 8 to 4 HPC registers physically available on the chip to monitor the applications' low-level behavior. It can be observed that reducing the number of HPC features from 8 to 4 has a relatively minor impact on the accuracy of most ML classifiers investigated for both malware and SCAs detection tasks. Therefore, to accordingly address the challenge of accurate runtime attacks detection, we adopt the four most prominent HPC features to make the proposed micro AI model applicable to most resource-limited edge platform architectures for both malware and SCAs detection. As the results show, JRip and J48 models are the two classifiers achieving higher detection accuracy $> 90\%$ for malware and benign classification. By comparison, all classifiers can achieve a high accuracy $> 90\%$ except NaiveBayes, highlighting that due to the higher complexity and variance, malware attacks require even a more careful analysis and selection of suitable HPCs and micro AI model to accurately distinguish the malicious patterns from normal traces. We also observe that SCAs detection accuracy is higher than malware detection accuracy with the same number HPCs and classification mode. Based on prior literature [9, 109], we can conclude that detecting malware is generally a more difficult task than microarchitectural side-channel attacks,

where [9] for SCAs gives close 100% accuracy while malware gives 85% accuracy in [109]. Malicious software attacks have continued to evolve in quantity and sophistication during the past decade. Due to the ever-increasing complexity of malware attacks and the financial motivations of attackers, malware trends are even recently shifting towards stealthy attacks by embedding the malicious code inside the benign application for harmful purposes that could simply bypass the standard detection mechanisms.

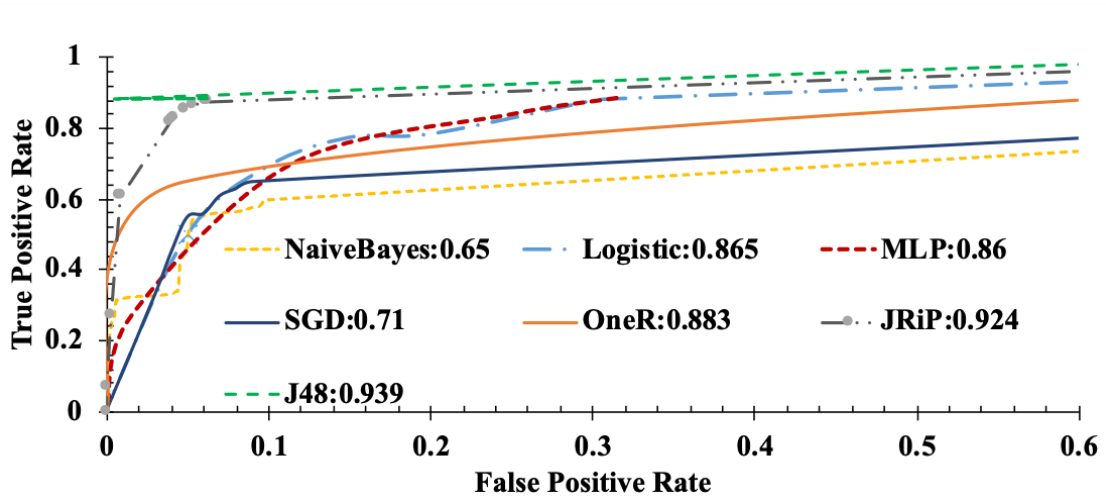
Table 4.9: False Positive Rate of various classifiers with 4 HPCs

Attacks	NaiveBayes	Logistic	MLP	SGD	JRiP	OneR	J48
Malware(%)	56.3	49.1	48.8	53	13.1	16.1	12.7
SCAs	5.6	14.9	5.3	23.8	0.9	1.3	1.3

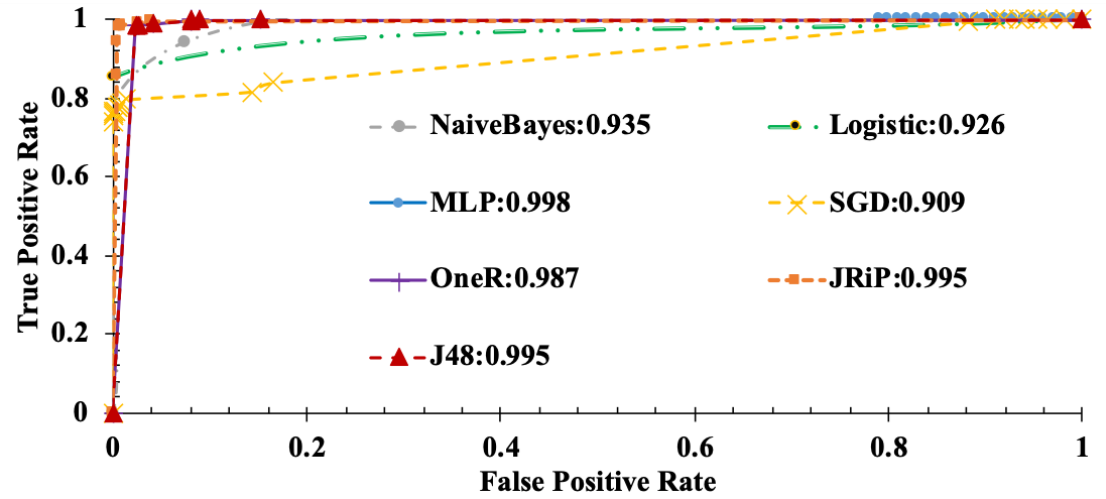
Table 4.10: F-measure of various classifiers with 4 HPCs

Attacks	NaiveBayes	Logistic	MLP	SGD	JRiP	OneR	J48
Malware	0.685	0.696	0.698	0.671	0.91	0.879	0.917
SCAs	0.929	0.919	0.97	0.876	0.989	0.981	0.987

- *False Positive Rate* Table 4.9 presents the false positive rate of all ML classifiers investigated in this work for malware and SCAs detection with 4 HPC features. For malware detection, we observe that JRiP, OneR, and J48 achieve lower FPR than the rest four classifiers with J48 delivering the lowest false positive rate of 12.7%. In addition, the proposed SCAs detection using various classifiers gives a much lower false positive rate as compared to malware detection where JRiP, OneR, and J48 achieve low false positive rates, 0.9%, 1.3%, and 1.3%, respectively. - *F-Measure* F-measure (F-score) is interpreted as a weighted average of the precision (p) and recall (r) which is formulated as $\frac{2\times(p\times r)}{p+r}$. The precision is the proportion of the sum of true positives versus the sum of positive instances, and the recall is the proportion of instances that are predicted positive of all the positive instances. F-measure can be considered as a more comprehensive evaluation metric over accuracy (percentage of correctly classified samples) since it accounts for both the precision and the recall values. More importantly, F-measure is also resilient to class imbalance in the dataset, which is the case in our experiments. Table 4.10 presents the F-measure results of all implemented ML



a) malware detection



b) SCAs detection

Figure 4.12: ROC Curve and AUC values comparison across various classifiers

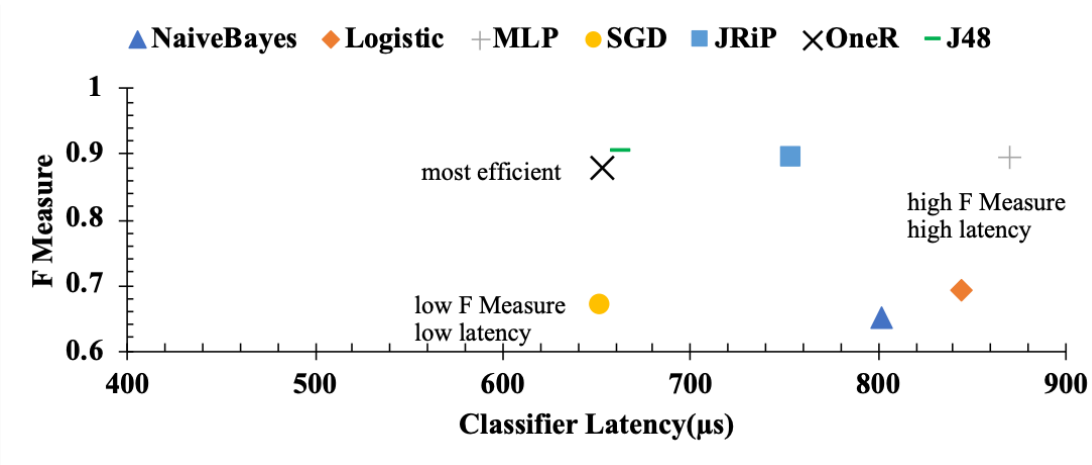
classifiers for malware and SCAs detection based on the top 4 HPC features. As seen, for malware detection, J48 model gives the highest F-measure value, 0.917, followed by JRiP, having a similar trend as accuracy. For the SCAs part, JRip gives the highest F-measure with 0.989 followed by J48. Hence, it can be concluded that J48 and JRiP are the two classifiers capable of achieving a high F-measure for both malware and SCAs detection.

- *Area under the ROC Curve (AUC)* Receiver Operating Characteristics (ROC) Curve plots the fraction of true positives rate versus the fraction of false positives for a binary classifier. The best possible classifier would thus yield a point in the upper left corner or coordinate (0,1) of the ROC space, representing 0% false positives and 100% true positives. The Area under the

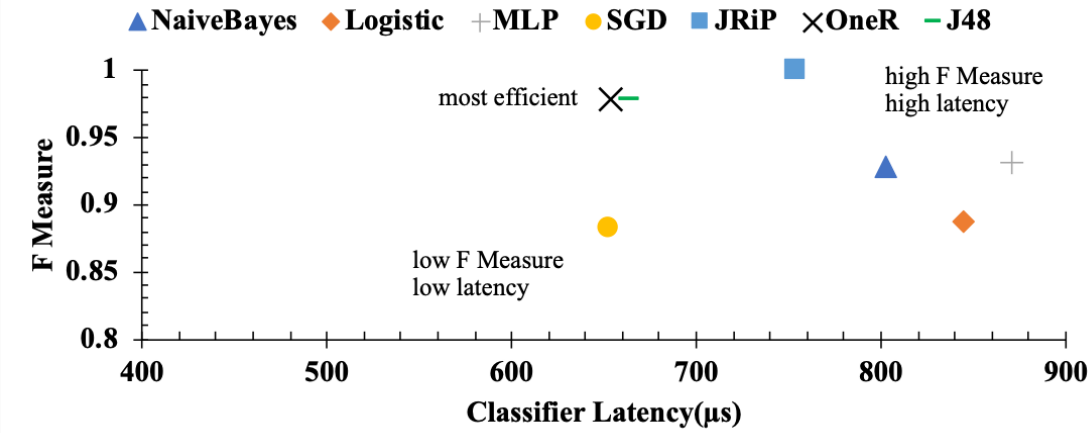
ROC Curve (AUC) metric corresponds to the probability of correctly identifying "under attack" and "under no attack" and robustness is referred to how well the classifier distinguishes between the two classes, for all possible threshold values. Higher AUC value represents better robustness for the ML-based micro AI countermeasure. Due to space limitation and given that NaiveBayes, Logistic, MLP, and SGD have resulted in lower detection accuracy for malware detection, their ROC curve and AUC values are not presented in Figure 4.12-(a). Similar to the F-measure rate, we notice in Figure 4.12-(a) that the OneR algorithm performs the worst in terms of ROC Curve for malware detection having the biggest distance to point (0,1) among the three other ML models. In comparison, the J48 classifier is closer to the coordinate (0,1) with a 0.939 AUC value, indicating a higher true positive rate and less false positive rate compared with the other seven ML classifiers evaluated in this work. Moreover, for SCAs detection, the ROC curve and AUC values of all seven classifiers are presented in Figure 4.12-(b). The results demonstrate that most classifiers with 4 HPCs are able to give a high AUC value and among them, JRiP, MLP, and J48, can yield above 0.99 AUC value.

- *Software Implementation and Efficiency Analysis* The software implementation and efficiency analysis are discussed in the section. As presented in Table 4.11, the seven classifiers discussed in prior sections are implemented at the Linux kernel level on the Intel I5 processor to evaluate the incurred software overheads. The overhead includes the time spent on reading the HPCs and the time spent on executing the classifiers. Since Intel processors are equipped with Turbo Boost technology, time measurement may suffer from time measurement error. Hence, we disabled Turbo Boost and set CPU to governor mode with 1200MHz operating frequency to avoid possible measurement errors. Performance counter reading overhead is negligible in kernel space when monitoring a single core, but overall system overhead increases for monitoring processes running on multiple cores, which may go up to as high as 20% depending on the frequency of events that are being sampled. The results for software implementation overhead of these classifiers show them to be slow with the execution time in the range of milliseconds, which is the order of magnitude higher than the latency needed to capture malware at runtime. It is important to note that several studied malware have execution time in the range of milliseconds or less, requiring fast detection to prevent them from corrupting the system.

Furthermore, we demonstrate the efficiency, i.e., F-measure vs. latency, of all the seven



a) malware detection



b) SCAs detection

Figure 4.13: Efficiency comparison across various classifiers with 4 HPCs

explored classifiers against malware and SCAs in Figure 4.13-(a) and Figure (b). A classifier with a higher ratio is considered a more efficient detector than the classifier with a lower ratio. As shown in Figure 4.13-(a) and (b), a clear trade-off is seen between detection rate and latency achievable for hardware-assisted malware and SCAs detection. For malware, J48 and OneR give high F-measure with 0.917 and 0.879 respectively, and incur the least computation overhead compared to others. Though JRiP gives a higher F-measure than OneR, it incurs over 100 *us* computation overhead. Similarly, ML classifiers such as MLP for SCAs achieve a high F-measure rate and higher computational overhead. The techniques such as SGD and Logistic show relatively more minor timing costs with low SCAs F-measure. While J48 and OneR having high F-measure with above 0.98 F-measure are more suitable for highly resource-constrained embedded systems due to

Table 4.11: ML classifier execution overhead

Classifier	Latency(ms)	Classifier	Latency(ms)
SGD	0.652	NaiveBayes	0.802
OneR	0.653	MLP	0.87
Logistic	0.844	JRIP	0.653
J48	0.663		

the smallest computational overhead. Clearly, the results show trade-offs between F-measure and latency. Therefore, it is crucial to compare ML classifiers for effective SCAs detection by considering all these parameters.

- *Hardware Implementation* As discussed, the software implementation of ML classifiers for malware and side-channel attacks detection is slow in the range of hundreds of microseconds which is an order of magnitude higher than the latency required to capture the malware at runtime, making it less a practical and effective solution for effective runtime detection of attacks. In this work, we implement the ML models on an FPGA to present the reduced latency and higher efficiency of hardware implementations compared to the software implementations. FPGA is a target in our study, as few modern microprocessors have provided access to on-chip FPGA units for programmable logic implementation. In recent years, SoC FPGA has been widely studied in both industry and academia [138, 139], which integrates both CPU processor and FPGA into one device. Such arrangement makes it feasible to implement reprogrammable low-level attack (e.g., malware and side-channel attacks) detection logic of the adopted ML models which can detect the malicious pattern by reading the CPU hardware performance counters through the shared memory bus. Under such design, the communication latency is relatively minimal as compared to the execution time of detecting malicious and benign programs and can be ignored. Hence, the total latency considered in our work is the time spent for ML-based detectors on the target FPGA for detecting the attacks and the communication latency was considered negligible as compared to hardware computational latency of the ML-based detectors. To address this challenge, in this work, we develop a hardware implementation of the adopted ML classifiers to analyze the efficiency of micro AI solutions for securing edge devices. To this aim, we use Xilinx Vivado Design suite to synthesize ML classifiers

for Xilinx Virtex 7 FPGA. Latency and power estimation are collected at 10ns clock cycle time. The accuracy of ML classifiers is based on data collected at 10ms intervals using perf.

Therefore, when it comes to adopting effective AI/ML models for hardware implementation in edge devices, the accuracy of an algorithm is not the only parameter in decision-making. This motivates the analysis of effective micro AI solutions for on-chip intelligence. In these methods, the design area and response time (latency) overheads of the ML classifiers also play a crucial role in selecting the cost-efficient hardware solution, particularly in edge devices considered emerging resource-limited computing systems. While complex algorithms such as MLP and Logsitic could provide higher detection accuracy, they would add considerable hardware overhead and implementation cost to the design. In addition, given their complexity, they can be slow in detecting malware. All these factors would make such heavyweight ML models less suitable for micro AI-based countermeasures for on-chip intelligence in edge devices.

Table 4.12 presents the hardware implementation costs for various ML classifiers used for hardware-assisted malware and side-channel attacks using the 8 and 4 most important HPCs. The latency unit is represented by the number of clock cycles (cycles @10 ns) required to classify the input vector. The unit for power consumption is Watt.

As seen, the NaiveBayes and MLP models, as expected, result in a significant area, power consumption, and latency overhead, as compared to other ML methods across both 8 and 4 HPC-based detectors. Moreover, JRip classifier delivers the most efficient implementation costs as compared to other algorithms. This highlights the cost-efficiency of JRip algorithm, particularly using 4 HPC features, for building an efficient micro AI countermeasure and on-device intelligence against emerging malware and SCAs. Clearly, the results show some trade-offs between accuracy, latency, and area overhead. Therefore, it is vital to compare classifiers by taking all of these parameters into consideration.

4.3 Conclusion

The ever-increasing complexity of modern computing systems especially embedded systems and Internet-of-Things (IoT) devices, has led to the growth of security vulnerabilities, making such systems appealing targets for increasingly sophisticated cyber-attacks. The limited computation and power resources further increase the difficulty of designing detection modules on edge devices.

Table 4.12: Hardware implementation results of ML-based micro AI countermeasures

Number of Features	Classifier	Latency(cycles@10ns)	Power(W)	Area(LUTs+FFs+DSPs)
8 HPCs	NaiveBayes	233	1.34	58177
	Logistic	68	0.63	13041
	JRIP	4	0.436	1504
	J48	9	0.436	1801
	SGD	34	0.444	2556
	OneR	1	0.324	1258
	MLP	302	1.03	36252
4 HPCs	NaiveBayes	90	1.12	56633
	Logistic	59	0.54	11815
	JRIP	2	0.35	156
	J48	3	0.34	584
	SGD	22	0.36	2466
	OneR	1	0.18	292
	MLP	102	0.84	25667

In this chapter, we first present the detection module in autonomous vehicles as a case study by investigating the hardware-level features and ML classification models. The reliance of autonomous driving systems on computer systems to sense and operate in the physical world has introduced novel security challenges at the hardware level that need to be explored in a systematic way. As a result, we propose a highly accurate ML-based detector trained with microarchitectural features against emerging side-channel attacks. And then, we extend the work and propose an accurate and cost-efficient micro AI-enabled countermeasure for securing emerging edge devices against emerging malware and side-channels attacks using processors' HPCs data. To this aim, we comprehensively explore the suitability of applying various types of ML classifiers for hardware-assisted malware and side-channel attacks detection by precisely comparing them in terms of detection accuracy, F-measure, AUC metric, latency, power consumption, and hardware overheads. Given the implementation cost of on-chip HPCs and their limited availability and accuracy, the results of this research will help the designers in making effective architectural decisions on the number and types of HPCs needed to implement in future architectures and to better realize the challenges of leveraging micro AI solutions at the hardware level to most effectively improve the performance of ML classifiers for detecting the malicious software.

Chapter 5

Side-Channels Mitigation via Randomization and Obfuscation

Cache hierarchy was designed to allow CPU cores to process instructions faster by bridging the significant latency gap between the main memory and processor. In addition, various cache replacement algorithms are proposed to predict future data and instructions to boost the performance of the computer systems. However, recently proposed cache-based Side-Channel Attacks (SCAs) have shown to effectively exploiting such a hierarchical cache design. The cache-based SCAs are exploiting the hardware vulnerabilities to steal secret information from users by observing cache access patterns of cryptographic applications and thus are emerging as a serious threat to the security of the computer systems. Prior works on mitigating the cache-based SCAs have mainly focused on cache partitioning techniques and/or randomization of mapping between main memory. However, such solutions though effective, require modification in the processor hardware which increases the complexity of architecture design and are not applicable to current as well as legacy architectures. In response, this chapter explores the feasibility of adjusting existing system and hardware settings to pollute SCAs' observations with no hardware redesign overhead for current as well as legacy architectures. Based on the finding and SCAs detection work in Chapter 3, we further design a detection-mitigation approach to effectively mitigate the impact of side-channel attacks on last-level caches with minor performance overhead.

5.1 Background

5.1.1 Cache Hierarchy

Main memory is an off-chip memory with large latency of accessing data compared to the performance of processing cores. As a result, cache subsystem have been designed to bridge the latency gap enhancing the processing speed. Since in this work, the Intel core-I5 processor is deployed as the experimental platform, here we show the cache hierarchy of this processor in Figure 5.1. As depicted, each core has two L1 caches, one for caching instruction and the other one for caching data with 32 KB size. Each L2 cache with 256 KB is dedicated to one core which are not shared among cores while last level cache (LLC) or L3 cache is shared among all processing cores. Due to the large data size of L3 cache, Intel CPUs divide LLC into several slices as shown in Figure 5.1 which can be accessed concurrently and each core is connected to their own slice while they still have access to others through a ring bus interconnection [120, 140]. In order to balance the load between different slices, Intel employs an unpublished hash function to decide which slice a physical address belongs to.

All caches are organized in fixed-size B bytes line (64 bytes in this work) which is also the unit of allocation. The Intel I5 architecture adopts set-associative as cache-memory placement strategy which means all cache is organized as S sets of W lines, called a W-way set-associative cache [1, 120]. As shown in Figure 5.2, the lowest-order bits of address are called the line offset and used to locate data in the cache line with size of $\log_2 B$; the set index is adjacent to line offset and the number of bits is decided by line size ($\log_2 S$; the highest order bits are used as a tag for each cache line and identify whether one of the cache lines of the W lines is a cache hit. After the Sandy Bridge microarchitecture, last level cache are sliced and each address is transferred to a "slice id" with the usage of an unpublished hash function [1, 120].

5.1.2 Prefetcher Functionality

To further reduce the large main memory access latency, prefetcher units are designed in modern microprocessors that are responsible for fetching the data (as well as instructions) that will be more likely accessed in near future and bringing them from off-chip main memory to the on-chip cache memory [141–145]. In addition, various prefetching policies have been proposed to achieve a

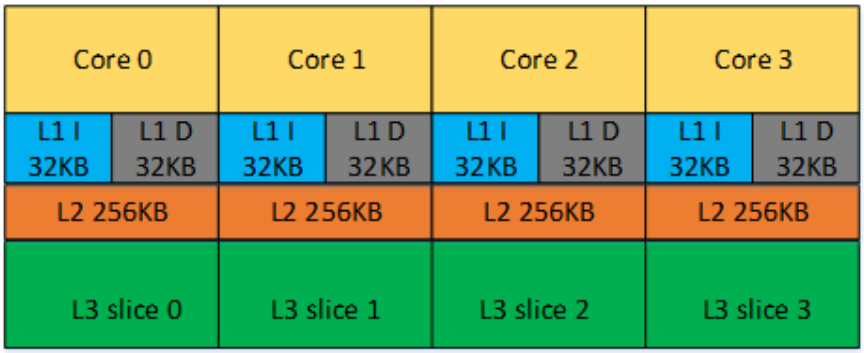


Figure 5.1: Cache architecture

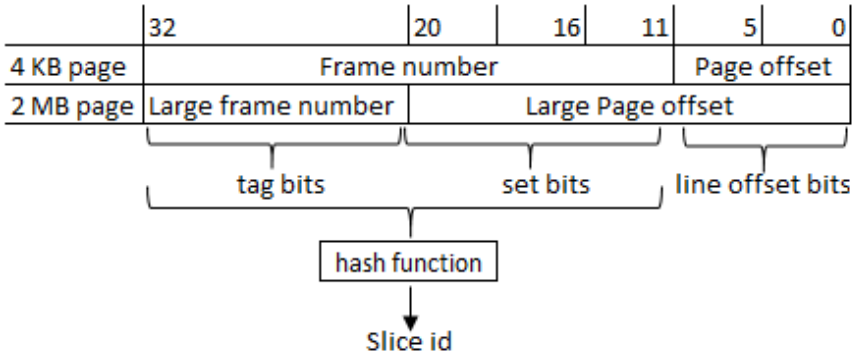


Figure 5.2: Address indexing [1]

Prefetcher	**Description**
DCU Hardware prefetcher	which fetches the next cache line into L1 data cache
DCU IP prefetcher	Uses sequential load history to determine whether to prefetch the next expected data into L1 cache from memory or L2
L2 hardware prefetcher	Fetches additional lines of code or data into the L2 cache
L2 adjacent cache line prefetcher	Fetches the cache line that comprises a cache line pair (128 bytes)

Table 5.1: 4 Prefetchers in Intel Computer Architecture

high performance and low prefetching cost, such as Stride [146] and GHB D/GC prefetching [144] schemes. As listed in Table 5.1 there exists four major prefectchers units in various Intel processor architectures such as Nehalem, Westmere, Sandy Bridge, Ivy Bridge, Haswell and Broadwell. DCU Prefetcher which is known as an streaming prefetcher unit fetches next cache line in advance from

lower cache hierarchy or main memory when multiple loads from the same cache line are detected and completed within a certain time period. Such design policy is suitable for regular data accessing. DCU Prefetcher stores former instruction pointer history and predicts the next data expected to be accessed, and then fetchesthe data from L2 cache or the main memory. On the other hands, L2 Hardware Prefetcher primarily monitors the pattern of data access and prefetches data at address X+2 from memory to L2 cache when data at address X and X+1 is requested. In addition, L2 Adjacent Cache Line Prefetcher is responsible for fetching the cache line to form a cache pair (128 bytes) from main memory. If this prefetcher is disabled, only one cache line (64 bytes) will be fetched when it is requested by the processor.

As shown in Table 5.1, there exist four prefetchers units in various Intel processor architectures such Nehalem, Westmere, Sandy Bridge, Ivy Bridge, Haswell, and Broadwell. On each core, there is a Model Specific Register (MSR) with the address of 0x1A4 that can be used to control the 4 prefetchers [147]. Bits 0-3 of the MSR are used to control functionality of the prefetchers. When the corresponding bit is set to 1, the prefetcher is disabled; otherwise, it is enabled. The value of the register can be changed either through the BIOS setting or directly writing a value to the register. In this work, we perform the latter method and change the functionality of prefetchers randomly during the execution of victim applications.

5.1.3 Prime+Probe Attack

The Prime+Probe attack contains two steps: 1) "prime": evicting cache sets that consist of victim's data with potential conflicting memory blocks; 2) "probe": accessing data of the memory blocks and measuring the access time. Compared to L1 Prime+Probe, L3 last level cache Prime+Probe attack is a more challenging side-channel attack due to the fact that L3 cache has a much larger size (6MB in this work) with higher latency compared to L1 cache, which makes the probing phase a more difficult and time-consuming process. Furthermore, current Intel processors divide the last level cache into different partitions each connected to different cores, hardening the recovery of mapping address by the attacker. Hence, L3 Prime+Probe attack firstly attempts to findpotential conflicting cache sets to narrow down the scope of Probing step and achieve high attack resolution, which is a critical step for successful attacks.For the "Probing" step, it randomizes eviction sets to eliminate the influence of prefetchers and re-access memory lines. Since in this work

we are focusing on Prime+Probe attack as case study, below we briefly describe the important steps for finding potential conflicting cache sets used in [1] which proposed and implemented L3 Prime+Probe.

- **Step 1.** Build a Large Page. Large page size can eliminate the need of address translation. Setup potential conflicting memory lines.
- **Step 2.** Expand. Iteratively add lines to the subset initialized as an empty subset as long as there is no self-eviction. Self-eviction is detected by priming a potential new member, accessing the current subset and timing another access to the potential new member.
- **Step 3.** Contract. Iteratively remove lines from the subset checking for self-eviction.
- **Step 4.** Collect. Scan original set, looking for members that conflict with the contracted subset.
- **Step 5.** Repeat until the original set is (almost) empty.

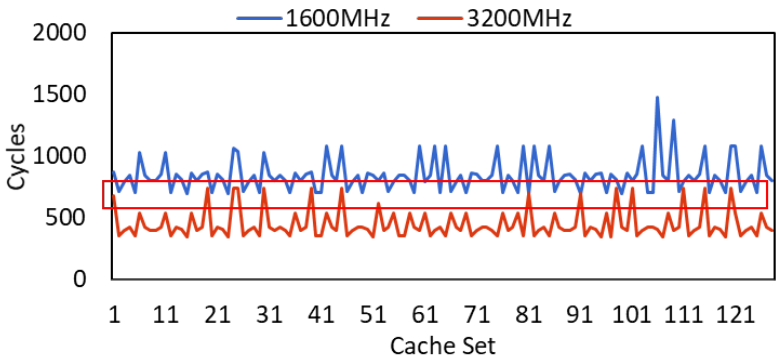


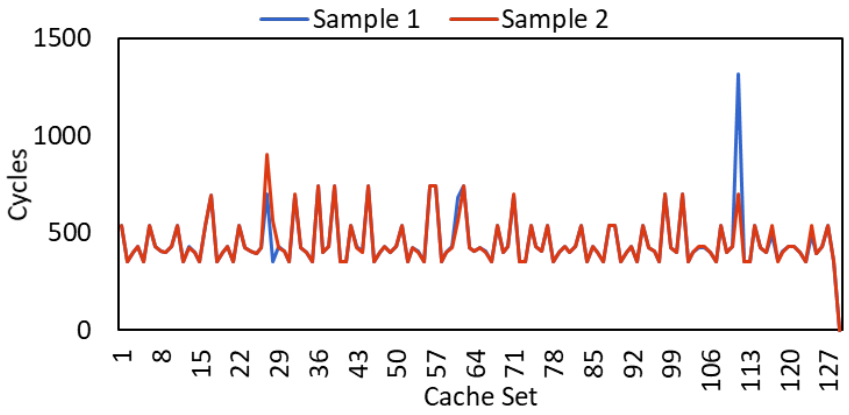
Figure 5.3: SCA probing time comparing 1600MHz and 3200MHz

5.1.4 Motivation

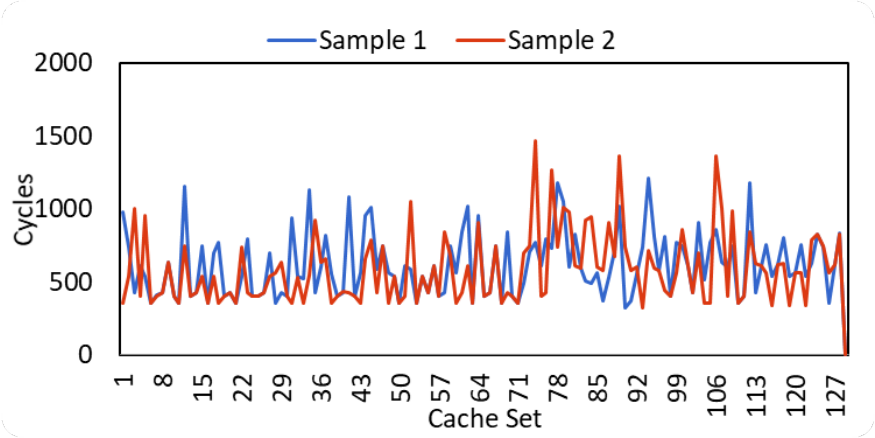
System Level Parameter Adaptation: Frequency

According to a recent study [1], the threshold should be set depending on the platform and frequency used, e.g., 700 cycles for the server under 2900 MHz and 400 cycles for the desktop under 3200 MHz. To highlight the influence of frequency scaling, the Probing results is plotted under 1600 MHz and 3200 MHz in Figure 5.3. In Figure 5.3, X-axis represents the cache set, and

Y-axis represents the accessing time in the unit of cycles. It observes that 700 cycles-per-accessing belongs to "long accessing time" (cache miss) under 3200MHz while it belongs to "short accessing time" (cache hit) under 1600MHz. It finds that the L3 cache hit threshold cannot separate the probing results highlighted in the rectangle region in the figure. Frequency randomization makes it difficult for the attacker to distinguish which cache set has been accessed.



a) Prefetchers under no adaptation



b) Prefetchers with adaptation

Figure 5.4: The impact of hardware prefectchers adaptation across two separate cache traces

Architecture Level Parameter Adaptation: Prefetchers

Prefetcher units are developed in modern microprocessors that are responsible for fetching the data/instructions predicted to be accessed in near future references [141–145]. As shown in Table 5.1, there are four prefetchers units implemented in various Intel processor architectures. The functionality of each prefetcher can be changed either through the BIOS setting or by directly writing

a value to the register [147]. In this work, we perform the latter method to randomize the functionality of prefetchers during run-time. Moreover, implementation of cache-based SCAs like L3 Prime+Probe leverages a randomizing function for accessing memory to avoid being influenced by prefetchers. However, such methods are effective only when prefetchers are enabled or disabled constantly.Thus, here we examine the impact of randomly enabling prefectchers in SCAs' observations.

In Figure 5.4-(a) and Figure 5.4-(b), X-axis represents the cache set, and Y-axis represents the accessing time in the unit of cycles. Prime Probe is executed twice, and two cache accessing traces are collected with the same frequency setting at 3200 MHz. As shown in 5.4-(a) without prefetchers' randomization, in most cases, the two samples' traces overlap with each other, indicating that the attacker can deduce cache pattern of victim applications with multiple cache traces and remove noise. Figure 5.4-(b) shows the probing results of two samples under prefetchers' randomization (when prefetchers are randomly enabled or disabled). It notices that two traces vary significantly over the accessing time, and no specific cache access pattern can be identified. Thus, randomly enabled prefetchers can add more noises to cache access traces that attacker observes.

5.2 Mitigating Cache-Based Side-Channel Attacks through Randomization: A Comprehensive System and Architecture Level Analysis

With increasing computation and performance demand, various components are proposed and deployed in processor architecture to boost performance, such as cache, branch predictor, out-of-order execution just to name a few. Despite the provided performance benefits, these solutions also causes new microarchitectural vulnerabilities which have been exploited by new type of attacks. Such Side-Channel Attacks (SCAs) observe side-channel information by causing interference and induce infer sensitive information and confidential data.

Timing-based cache side-channel attacks [1–3, 21, 113, 114] exploit the accessing time gap between the on-chip caches and main memory, and collect cache hit/miss traces based on various accessing times. By analyzing the traces, the attacks can infer sensitive information according to cache traces and the knowledge captured from the cryptographic algorithm. For instance, Flush+Reload side-channel attacks highly relies on the assumption that the victim and the attacker

share the same memory space and utilizes the cache-access timing information to retrieve the secret key from the system. Attacks such as Prime+Probe [1] supersedes the Flush+Reload attack and does not require any shared memory space with the victim to extract sensitive information. Due to the invisibility, feasibility and capability to expose and extract the secret keys in the cache-based SCAs, there is an urgent need to address the security risks posed by such attacks in present computer systems as well as legacy systems.

To cope with the challenges introduced by the cache-based SCAs, prior works have proposed various mitigation approaches to enhance the security and protect computer systems against side-channel attacks and information leakage. Some researches propose techniques to design software to avoid sensitive information leakage [52, 148–150]. In these techniques, by unifying control flow or bitsliced implementation of cryptographic applications, they can protect sensitive applications from being observed by side-channel information. However, general applications are still under SCAs threats. What's more, the microarchitectural details and Instruction-Set Architecture (ISA) implementations are a significant obstacles for software developers. Hence relying on software developers adopt new techniques to eliminate side-channel information leakage is not practical.

Some other researches have proposed architecture-dependent solutions to mitigate the side-channel attacks and protect the computer systems in which they are mainly relied on designing new architecture or modify the existing architecture. Such techniques consist of two major directions including 1) Cache partitioning [22, 24, 53] and 2) randomizing memory and cache mapping [23] The first approach is based on dividing the entire cache sets into different blocks which could potentially prevent the cache-based SCAs from interfering the victim applications. As a result, attacks do not have access to cache pattern of observing users' applications. The latter approach is based on randomizing cache placement and access patterns [25, 26] that attempts to make the mapping between memory addresses to cache indices unpredictable, hindering attacks from stealing information. However, it is notable that these SCA mitigation methods are mostly proposed as an architectural level solution which require design modifications to protect future architectures, and they are not applicable to current as well as legacy architectures since they require hardware redesign efforts. This highlights the importance of proposing an efficient and low-cost mitigation methodology which does not require any new hardware design or modifications and also can be applied to general-purpose applications without the need to alter the existing software.

In response to these challenges, in this work we propose a novel and lightweight randomization-based SCA mitigation methodology which is based on the knowledge of two important characteristics of cache-based side-channel attacks including a) accessing time to determine cache hit/miss; b) obtaining cache pattern with limited noises according to cache hit/miss. To this aim, we propose a comprehensive system (CPU frequency) and architecture (prefetchers) level randomization methodology to efficiently mitigate the impact of side-channel attacks on last-level cache eliminating the need to modify the cache architecture and no requirement for hardware redesign effort. The rationale behind our proposed solution is that the CPU frequency can influence accessing time and hardware prefetchers can cause cache eviction that could potentially add noise in victims' cache pattern. To this aim, by carefully adapting the processor frequency and prefetchers operation and adding proper level of noise to the attackers' cache observations, we attempt to change the attackers' observation from victim application's cache access pattern and protect the critical information from being leaked. Our proposed randomization methodology indicates that scaling frequency can change the attackers observation by changing the cache accessing time, indicating the possibility of hiding victim cache trace and protecting victims' information from being leaked. In order to show the effectiveness of our proposed mitigation methodology, L3 Prime+Probe [1] is employed as a case study since this attack does not require shared processor core or memory, and poses a greater threat compared to other cache-based SCAs.

5.2.1 Proposed Methodology

In this section, we present the details of the proposed randomization methodology for mitigating the cache-based side-channel attacks by adapting the frequency and prefetchers in modern computer systems as well as legacy systems.

Hardware Platform

In this work, all experiments are conducted on an Intel I5-3470 processor with Ubuntu 16.0.4 LST operating system with Linux kernel 4.13. In Intel I5-3470 on-chip cache memory subsystem, while L1 and L2 caches are exclusively separated, and L3 cache memory is inclusive and shared among all cores meaning that flushing out the data in the last level cache could also remove the data in L1. The inclusiveness of L3 cache provides a potential vulnerability for LLC attacks

Table 5.2: Hardware Platform

Processor	Intel I5-3470 CPU, single socket-4 cores
Frequency	1.6-3.2GHz
L1i Cache	32KB
L1d Cache	32KB
L2 Cache	256
L3 Cache	6144KB
Memory Capacity	8GB DDR3

to be exploited. The details of cache hierarchy are given in Table 5.2. Following, we will further introduce the scenarios for different frequency scaling and prefetchers activation settings considered in our experimental setup as shown in Table 5.1.

Table 5.3: Experiment Scenarios

Scenario	Frequency	Prefetchers
A	3200MHz	All Enabled
B1	1600∼3200MHz	All prefetchers Enabled
B2	2200∼3200MHz	All prefetchers Enabled
B3	2600∼3200MHz	All prefetchers Enabled
C1	3200MHz	DCU prefetcher Enabled
C2	3200MHz	DCU IP prefetcher Enabled
C3	3200MHz	L2 hardware prefetcher Enabled
C4	3200MHz	L2 adjacent cache line prefetcher Enabled
D	2600∼3200MHz	DCU IP prefetcher Enabled

Randomization Scenarios

Here, we introduce the studied scenarios for exploring the impact of randomizing prefetchers as an architecture-level and processor frequency as a system-level parameter on securing the

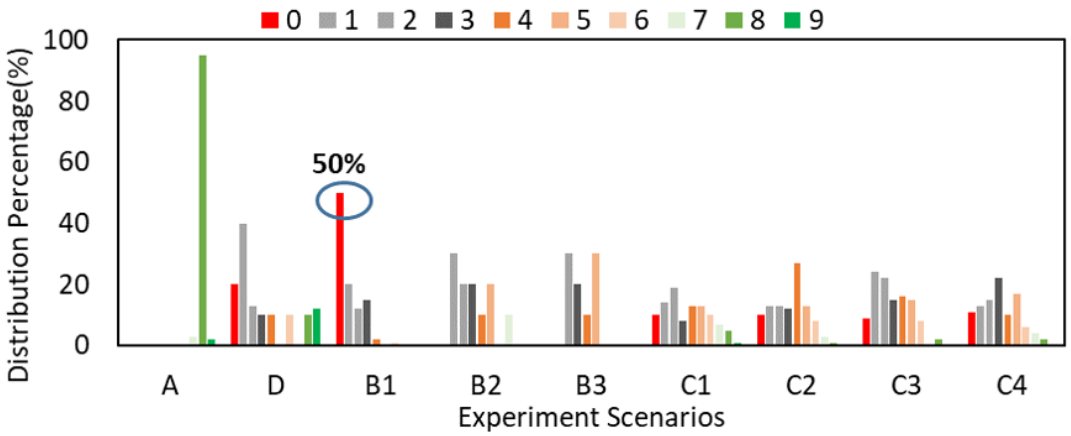


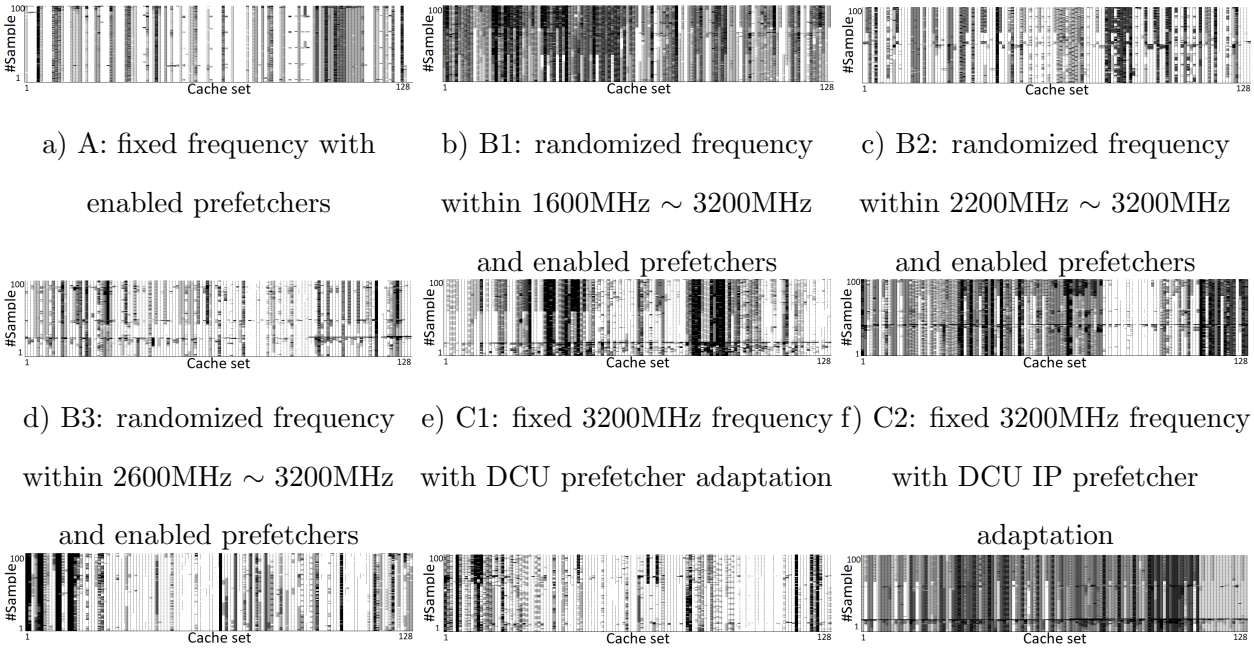
Figure 5.5: Group size of potential eviction sets (ranging from 0∼9) comparisons among A, B, C and D experiment scenarios

computer system against SCAs. By accounting for various randomization combinations, we attempt to conduct L3 Prime+Probe as a case study to collect observable cache patterns in which the attacker tries to obtain and extract sensitive information. In this work, the cache patterns will be employed to analyze the effectiveness of the proposed methodology. In addition, all experimented settings are listed in Table 5.3.

Scenario A: In order to obtain cache pattern information with less noise, Prime+Probe fixes frequency and avoids fluctuation in accessing cycles resulted from different frequencies. Hence, in scenario A we fix frequency to 3200MHz and enable all prefetchers. Scenario A is used as a comparison baseline with other randomization cases.

Scenario B1∼B3: Scaling frequency can change accessing time used by attackers to determine whether cache sets are accessed by the victim or not. Since scaling frequency changes applications' execution time (performance), we choose three ranges to evaluate the influence of scaling frequency values on victim's cache pattern and performance including 1600MHz ∼ 3200MHz, 2200MHz ∼ 3200MHz, and 2600MHz ∼ 3200MHz as listed in Table 5.3. It is notable that the larger the range is, the slower the performance becomes, since applications will be executed under lower frequency.

Scenario C1-C4: As mentioned, the Intel I5 cores have 4 prefetchers that can be enabled or disabled by writing the value to the memory address of 0x1A4 [120]. In these experiments, each of the four studied scenarios only enables one prefetcher for a random interval. Under such scenarios,



a) A: fixed frequency with enabled prefetchers

b) B1: randomized frequency within 1600MHz ∼ 3200MHz and enabled prefetchers

c) B2: randomized frequency within 2200MHz ∼ 3200MHz and enabled prefetchers

d) B3: randomized frequency within 2600MHz ∼ 3200MHz and enabled prefetchers

e) C1: fixed 3200MHz frequency with DCU prefetcher adaptation

f) C2: fixed 3200MHz frequency with DCU IP prefetcher adaptation

g) C3: fixed 3200MHz frequency with L2 hardware prefetcher adaptation

h) C4: fixed 3200MHz frequency with L2 adjacent prefetcher adaptation

i) D: randomized frequency within 2600MHz ∼ 3200MHz with C2

Figure 5.6: AES cache access heatmap under different randomization cases (A, B, C and D)

the prefetched data will pollute attacker observations since some data will be evicted due to the prefetching process. To this aim, four scenarios are devised to evaluate the effectiveness of hiding the victim's cache pattern for different prefetchers.

Scenario D: In this scenario which is considered as the combined scenario for adapting frequency and prefetchers, the least frequency scaling range and the most effective prefetcher to hide victims' cache pattern are tuned concurrently. The prefetcher size is chosen based on the result of C1 ∼ C4.

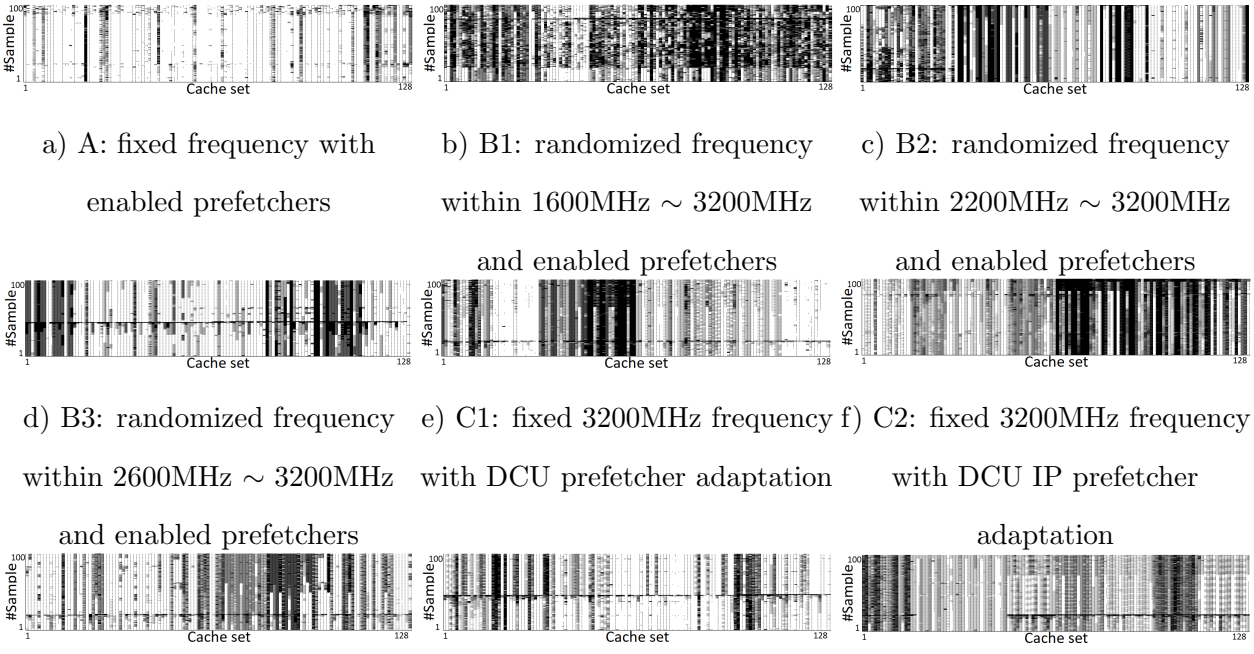
Experimental Methodology

As mentioned before, last level cache Prime+Probe attack is used as a case study to evaluate the effectiveness of our proposed approach. Due to the behavior of L3 Prime+Probe attack, here we evaluate the randomization influence based on two important factors including the size of eviction sets and probing results. To this aim, two different experimental methodologies are adopted and detailed below: - *Eviction Set-based Analysis* For thoroughly analyzing the eviction sets results, we have considered three different applications running at the same time including a) victim; b)

attack, where the potential eviction sets are built first and then probing the potential eviction sets takes place; and c) applying various randomization scenarios for SCA mitigation (discussed in Section 5.2.1). As a result, eviction sets building process will be under randomization influence and the group size of eviction sets can be successfully collected. - *Probing-based Analysis* As mentioned earlier, influence on eviction sets needs to be removed to obtain the impact of probing results. For this purpose, in our comprehensive analysis, various randomization scenarios begin after building the eviction sets step, and victim and attack applications are executed concurrently. As a result, once the attacker prepares the eviction sets, probing and randomization scenarios start concurrently.

5.2.2 Experimental Results and Evaluation

In this section, we present the experimental results and evaluation analysis of the proposed randomization-based SCA mitigation methodology. As described in the proposed methodology, we extract eviction set size and probing phase results of scenario A, and compare it with randomization cases (B, C, and D) to effectively analyze the level of noise added in attackers' observed information.



a) A: fixed frequency with enabled prefetchers

b) B1: randomized frequency within 1600MHz ∼ 3200MHz and enabled prefetchers

c) B2: randomized frequency within 2200MHz ∼ 3200MHz and enabled prefetchers

d) B3: randomized frequency within 2600MHz ∼ 3200MHz and enabled prefetchers

e) C1: fixed 3200MHz frequency with DCU prefetcher adaptation

f) C2: fixed 3200MHz frequency with DCU IP prefetcher adaptation

g) C3: fixed 3200MHz frequency with L2 hardware prefetcher adaptation

h) C4: fixed 3200MHz frequency with L2 adjacent prefetcher adaptation

i) D: randomized frequency within 2600MHz ∼ 3200MHz with C2

Figure 5.7: RSA cache access heatmap under different randomization cases (A, B, C and D)

Eviction Sets-based Randomization

AES and RSA applications under L3 Prime+Probe are executed under all nine scenarios and different group sizes of eviction cache sets are collected. In order to effectively avoid from possible noises in the results, the attack is executed one hundred times. Figure 5.5, depicts the eviction sets results ranging from 0 to 9 across various randomization scenarios. As shown, X-axis represents different scenarios and Y-axis represents the number of eviction sets group size.
- *Frequency Randomization Analysis* For scenario B1 under frequency randomization, the result depicted in Figure 5.5 shows that there is a 50% possibility that the attacker builds 0 eviction set, indicating that the attacker is not able to find eviction sets and the attack will not be successful. Furthermore, a large percentage (above 40%) of samples group size is below or equal to 3, indicating a high possibility of missing the real conflict cache set. However, reducing scaling range (B2 and B3) decreases the effectiveness of causing the failure of building eviction sets. This fact indicates that scaling frequency with large range can confuse cache hit/miss during the time that attacker application is building eviction sets. For all three frequency scaling scenarios, eviction sets are all lower than scenario A. There exist two main reasons for this phenomenon. First, scaling frequency withholds the execution of attackers programs making some of the attackers' cache sets to be evicted due to frequency scaling. Secondly, scaling frequency changes the execution performance of both victims and attackers instructions, which results in the wrong conflict eviction set found in Step 2, Expand. As a result, once the attacker expands the cache sets by re-accessing the cache sets to find the conflicts, this process can misguide the attacker's observation in identifying the potential victim sets to comprise the security of the system.

- *Prefetchers Adaptation Analysis* Similarly, Figure 5.5 compares eviction group size of under C1∼C4 scenarios. As can be seen, nearly 10% of samples are 0 meaning that no potential eviction set is found and the attack can not proceed. It can also be observed that the group sizes of 100 samples are evenly distributed across 1∼7 eviction sizes. This will result in two observations that affect the success of attacks in leaking information. First, the attacker can miss the actual conflict cache lines. Second, the attacker is not able to collect the aligned cache traces each time while L3 Prime+Probe requires 300∼1000 sample traces [1]. Furthermore, it is notable that different prefetchers components have a relatively similar effect on evictions sets building results. Our comprehensive analysis shows that compared with frequency scaling, adapting prefetchers is less

effective in preventing attackers to identify potential conflicts for mitigating the attacks. This is because prefetchers can only change instructions' execution of victim and attack applications running on the system when requested data is being fetched in advance. That being said, prefetchers adaptation still shows high effectiveness for causing a disturbance in attackers observation.

- *Analysis of Concurrent Adaption of Frequency and Prefetchers* Scenario D contains both frequency scaling and prefetchers adaptation. As shown in figure 5.5, the distribution shows a more similar trend to the one observed in scenario C in which around 20% of the SCA attacks are failed due to the failure of building potential conflict sets.

Probing Results

According to the behavior of L3 Prime+Probe attack, we probe the potential conflicting cache set (eviction cache set) and use heatmap analysis to examine the victim applications' cache accessing pattern. RSA and AES are used as victim applications. To show the probing results which can not be separated with the threshold, the heatmaps are plotted with 400 cycles suggested in [1] for the hit/miss threshold. As shown in Figure 5.6 and Figure 5.7, black blocks represent cache misses, meaning that victim accessed the cache set; white blocks represent cache hit that corresponds to the case in which the victim did not access the cache set. Hence, the attackers need to obtain more clear black blocks to find out the cache access pattern of the victim application for leaking information.

- *Frequency Randomization Analysis* As shown in the results, as compared to scenario A, both Figure 5.6-b), c) and d) and Figure 5.7-b), c) and d) show that the trace of victim has been significantly polluted with cache misses resulted by the frequency randomization. Furthermore, it can be observed that the larger the gap of frequency scaling range is, the higher noise the victims' cache trace has. Comparing B1, B2 and B3 in both figures depicts that scaling frequency from 1600MHz to 3200MHz can obtain higher noise and hide victim cache access pattern more efficiently. Such a phenomenon is because scaling frequency can change cache accessing time and mislead attackers leading them to identify cache hit/miss incorrectly. In addition, by increasing the gap, cache hit under low frequency and cache miss under high frequency are more likely to overlap, making cache traces contaminated. In all three frequency randomization scenarios (B1, B2, and B3) for both AES and RSA applications, no victim cache access pattern can be found, indicating that

the attackers are not able to infer victim's secret information.

- *Prefetchers Adaption Analysis* Both Figure 5.6 and Figure 5.7 illustrate cache traces under different prefetcher settings. Generally, our comprehensive analysis across various configurations and scenarios indicates that randomly enabling/disabling different prefetchers (C1∼C4) results in less contamination of cache traces compared to frequency scaling. This is due to the fact that prefetcher units can only evict cache sets while scaling frequency influences all cache sets accessing time. Among all prefetchers, it can be found that DCU IP prefetchers adaptation (C2) is the most effective one compared to the other scenarios. Hence, in our optimal case study (Scenario D), DCU IP prefetcher is chosen as the target prefetcher unit to be tuned concurrently with frequency.

- *Concurrent Adaption of Frequency and Prefetchers* As can be observed from the results, for both AES and RSA benchmarks, cache traces resulted from optimal scenario of D in which both frequency and prefetcher units are selected at their optimal values, the concurrent adaptation shows more efficiency than the conditions of scaling frequency or adapting DCU IP prefetcher solely. The pollution extent of D is similar to the B1 scenario, indicating scaling frequency with a smaller range by adding prefetcher adaption.

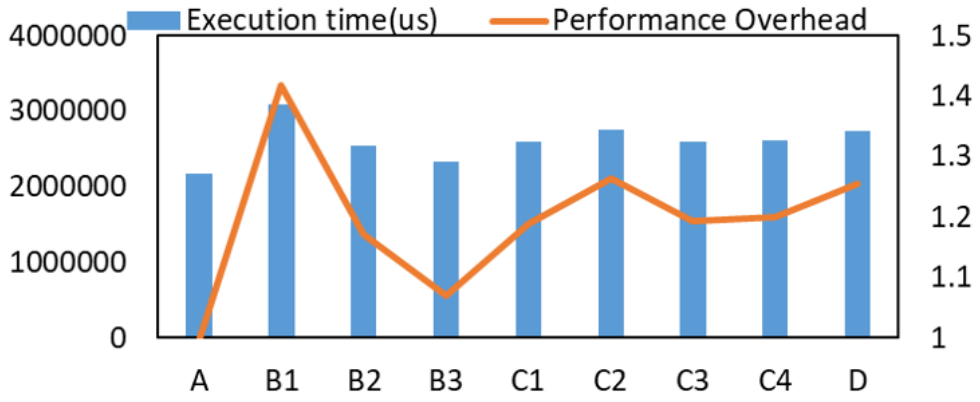


Figure 5.8: A ∼ D execution time and performance overhead where A ∼ D is normalized by A

Scenario	A	B1	B2	B3	C1	C2	C3	C4	D
Error Rate	6%	17%	41%	53%	32%	37%	29%	30%	58%

Table 5.4: Error Rate of Flush+Reload recovered key

Flush+Reload Analysis and Discussion

In addition to Prime+Probe, we also test the proposed randomization strategy on Flush+Reload side-channel attack and evaluate the effectiveness of randomization for mitigating shared memory based SCAs.

Flush-Reload [2, 74, 75] exploits the weakness of page de-duplication and monitors memory access lines in shared memory pages. This attack flushes out the victim data in the cache and waits for the victim application execution. The attacker then reloads data by accessing them and measures the accessing time. If accessing time is shorter, it infers the data has been accessed by the victim, otherwise, it is concluded that the data has not been accessed by the victim application. Since Flush+Reload already has the knowledge of victim's memory access, it only monitors the certain position in the cache. In our experiments, we have considered reduce, divide, and multiply as three operations under spy for Flush+Reload attacks RSA. Hence, there is no building eviction sets and probing parts. Due to space limitation, here we only report the error rate results to evaluate the effectiveness of our proposed mitigation methodology on the Flush-Reload attack. As shown in Table 5.4, the error rate of recovered keys under randomization (Scenarios B/C/D) increases significantly compared to the obtained error rate under Scenario D. This observation indicates the effectiveness of proposed methodology to protect system from Flush+Reload SCA. In addition, it is noticeable that the randomization Scenario D with the error rate of 58% outperforms all other scenarios in terms of error rate. Overall, since Prime+Probe and Flush+Reload represent non-shared memory cache-based SCA and shared memory cache-based SCA, respectively, it is safe to conclude that the proposed randomization methodology in this work can be further applied to other cache-based SCAs, like Flush+Flush, etc. with minimal modification and overhead.

Performance Overhead Analysis

In this section, we choose MiBench [82] benchmark suites to evaluate the performance overhead of applications caused by randomizing frequency and prefetchers in terms of application execution time. In order to eliminate the influence of random noise on execution time caused by system scheduling, etc., each application from the benchmark under the eight scenarios is executed 100 times to avoid the interference of random noises. Figure 5.13 shows the arithmetic average execution time and performance overhead where A to D settings are normalized by the execution

time of A.

From Figure 5.13, it can be observed that scaling frequency with largest range (Scenario B1) causes highest performance overhead (around 40%) compared to the lowest overhead case study which is Scenario A. As shown in Figure 5.13, by reducing the frequency scaling range (Scenarios B2 and B3), the performance overhead decreases to 23% and 18% highlighting the effectiveness of adapting frequency in lower range for efficient SCA detection and mitigation. Another interesting observation is that the DCU IP prefetcher in Scenario C2 has shown more influential performance reduction to applications' performance as compared to three other Scenarios (C1, C3, and C4). As depicted, under C2, performance overhead is 27% while the remaining three prefetcher adaptation scenarios achieve nearly 20% overhead. On the other hand, Scenario D has shown slightly higher overhead than adapting DCU IP prefetcher (C2) because applications are executed in high (3200MHz) and low frequency (2600MHz) with substantial frequency gap in between. Compared to closest state-of-the-art work [54], our proposed randomization methodology in this work based on frequency scaling and prefetchers adaptation, achieves significantly lower performance overhead, reducing the overhead from 32.66% to around 20% of performance loss. Furthermore, the proposed randomization-based solution can be effectively adopted only when victim applications are executed which are the target of side-channel attacks.

Table 5.5: Comparison of the recent works on SCAs protection (Red color indicates drawbacks of the work)

Prior Works	Detection Metrics				Mitigation Metrics		
	HPCs Evaluation	Required SCAs' Code	Latency	False Alarm Solution	New Architecture	Performance Overhead	Application Modification
CloudRadar [9]	Yes	Yes	100s-5000s	No	No mitigation module		
Real-Time Detection [151]	No	Yes	1.5 2.4s	No	No mitigation module		
Nights-watch [17]	No	No	No Mentioned	No	No mitigation module		
Cacheshield [19]	No	No	11 ms-24ms	No	No mitigation module		
CPU Elasticity [54]	No detection module				No	32.66%	No
FLUSH+PREFETCH [152]	No detection module				No	Not Mentioned	No
Random Fill [26]	No detection module				Yes	No Mentioned	No
Catalyst [22]	No detection module				Limited Architectures	0.16%-17.6%	No
★ Hybrid-Shield	Yes	No	500 μs–5ms	Yes	No	15%	No

5.3 Accurate and Efficient Cross-Layer Countermeasure for Run-Time Detection and Mitigation of Cache-Based Side-Channel Attacks

In this chapter, Section 5.2 shows the effectiveness of randomizing the settings of frequency and hardware prefetchers for polluting the SCAs' pollution. However, the high performance overhead incurred by frequency adjustment and prefetchers' adaptation limit the deployment of the light-weight mitigation approach. In response to the challenges above, this work proposes the *Hybrid-Shield*, an accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. For the detection stage, microarchitectural information of victim under attack and under no attack conditions is collected for training and testing ML classifiers. The ML-based detectors are then deployed to facilitate an effective run-time SCAs detection using only the HPCs features of victim application with high prediction accuracy and low instance level false alarm rate. For the mitigation stage, *Hybrid-Shield* offers a lightweight system and architecture level randomization technique to effectively mitigate the security threat from SCAs with no hardware redesign overhead. To this aim, by carefully adapting the processor frequency and prefetchers, a proper level of noise is added to the attackers' cache observations, protecting the critical information from being leaked. Table 6.1 characterizes the existing countermeasure techniques to combat the challenges of SCAs and further highlights the contributions of our proposed work. *Hybrid-Shield* eliminates the need for side-channel attacks' HPCs information, supports an effective false alarm minimization method, and does not require additional architecture modification for SCAs mitigation with less performance overhead.

Contributions. The main contributions of this work are summarized as follows:

- To eliminate the influence of untrustworthy attack's HPCs, *Hybrid-Shield* first attempts to detect SCAs based on only victim applications under two conditions: 1) Victim under Attack (VA), and 2) Victim under No Attack (VNA).

- Various ML classification algorithms are explored to find the most accurate classifier for detecting SCAs.

- A customized set of HPC features and False Alarm Minimization (FAM) are proposed to

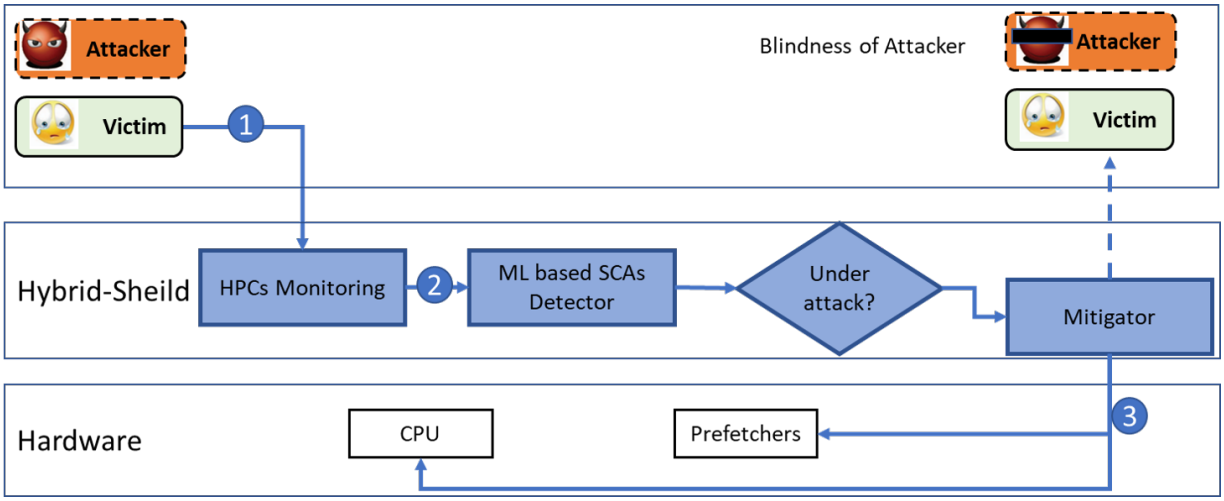


Figure 5.9: Overview of Hybrid-Shield

improve the detection accuracy while lowering the false alarm rate.

- *Hybrid-Shield* is further equipped with a comprehensive system and architecture level randomization solution to mitigate the SCAs on last-level caches efficiently.

5.3.1 Proposed Methodology

To combat the cache-based SCAs, *Hybrid-Shield* includes a run-time detector that identifies the SCAs in the system. It then activates a mitigator module after receiving an attack alarm by randomizing a system (frequency) and adapting architecture level (prefetchers) parameters. In particular, each victim application is executed on a fixed core and HPCs are collected when the victim is running, as shown as Step 1 in Figure 5.9. Next, as depicted in Step 2 in Figure 5.9, the collected HPC data is fed to a trained ML based detector which decides if the victim application is under attack. If the detection result is "under attack", then as shown as Step 3 in Figure 5.9, the mitigator randomizes the settings of frequency and prefetchers to mitigate the impact of the detected SCA. Such randomization-based adjustment contaminates the cache traces and makes attacks effectively blind.

Threat Model

As for the threat model in our experiments, we consider multi-core computing environments that use the inclusive (L3 cache) and the non-inclusive cache memories (L1/L2 cache) on the Intel

architecture, in which benign and malicious processes deploy shared libraries. It is assumed that the environment is Linux-based single OS environment, and both victim and attack applications reside in the same physical machine, either running on the same core or different processing core. The system could face side-channel attacks comprising of shared-memory based attacks (e.g. Flush+Reload), and also none shared-memory based attacks (e.g. Prime+Probe). It is further assumed that the potential SCAs might be hidden inside benign application or be crafted to emulate the benign application behavior. In addition, *Hybrid-Shield* attempts to detect the SCAs based on only victim applications under two conditions: 1) Victim under Attack (VA), and 2) Victim under No Attack (VNA).

Hybrid SCAs Detection & Mitigation vs. Stand-alone Mitigation

To highlight the importance of randomizing both system-level setting (frequency) and hardware-level setting (prefetchers) concurrently, here we examine the performance overhead of stand-alone randomization-based mitigation. Taking RSA application under the Prime+Probe attack as an illustrative case study, six randomization scenarios are listed in Table 5.3. The corresponding cache patterns under different randomizing scenarios are shown in Figure 5.7. The black blocks represent the cache accessed by RSA. Hence, observing the clear black blocks patterns can help the attacker to extract users' information. Applying system and/or architecture level randomization increases the number of black blocks (accessed cache blocks), leading to substantial contamination of observed cache access patterns by the attacker. Also, as seen, randomization could mask the real victims' cache patterns. The performance overhead of various randomization scenarios is shown in Figure 5.13. One can observe that scaling frequency and adapting prefetchers with various settings yield in different protection levels with various performance overheads. Also, the randomization performance overhead could increase to as high as 40% (scenario B1). To cope with the high performance overhead, we propose to balance the need to eliminate side-channel information leakage and improve performance.

Randomization Evaluation Criteria

To determine comprehensive evaluation criteria for selecting the optimal randomization strategy, *Hybrid-Shield* accounts for both error rate and performance overhead evaluation metrics.

The attack protection effectiveness represented by *ProtectionLevel* is calculated by normalizing the number of bits correctly recovered from cache traces under randomization with the number of bits recovered under no randomization. The higher the *ProtectionLevel* is, the more secure system is. In addition, the performance overhead referred to as *Overhead* indicates the total execution time under the applied mitigation strategy, which is normalized by the execution time under no mitigation strategy. As a result, we model the randomization evaluation metric in *Hybrid-Shield*. To examine the effectiveness and performance cost our randomization-based mitigation approach uses Equation 5.1 described below:

$$S = (\alpha * 1/ProtectionLevel) + (\beta * Overhead) \tag{5.1}$$

where S denotes the overall randomization score, α and β are the error rate and performance overhead, respectively. One can observe that the larger $ErrRate$ provides a higher level of security and protection against SCAs, and the smaller $Overhead$ indicates the less performance cost of the mitigation scenario. Hence, in our proposed randomization strategy described in Algorithm 1, we target to minimize the overall randomization score (S) to find out the most efficient system and architecture level randomization scenario.

Randomization Strategy Selection

In this subsection, we present the details of the proposed randomization strategy selection in *Hybrid-Shield*. There exist four settings of prefetchers and frequencies on our studied Intel processor system. We consider two sets of adjustments for each hardware prefetcher, including "1" as "deactivated" and "0" as "activated", while the frequency is ranged from 1600 MHz to 3200 MHz. Assuming that frequency value is $1600 + N * Step(n >= 0)$, then the number of possible frequency values is N. Hence, the number of prefetchers and frequency settings is $2 * 2 * 2 * 2 * N$. In this work, we set the initial value of Step as 400 MHz, corresponding to N=4. Moreover, the number of transitions from one setting to another setting is calculated: $2 * 2 * 2 * 2 * N = 16 * 4 = 64$. To further reduce the number of settings required to iterate, we propose a randomization strategy selection process shown below which helps in decreasing the number of the needed randomization settings search to 8∼16.

Step 1. Execute victim application with highest fixed frequency and enabled prefetchers. Collect execution time.

Step 2. Execute victim application with SCAs and recover key.

Step 3. Pick a random randomization strategy S_i among the 64 choices. Collect the execution time of victim under the strategy as Ei and key recovered as Ki.

Step 4. Choosing the strategy S_j with lower frequency while prefetchers' setting remain same as S_i. Collect the execution time of victim under the strategy as E_j and key recovered as K_j.

Step 5. Choosing the strategy $S_j + 1$ with more prefetchers are disabled while frequency remains same as S_i. Collect the execution time of victim under the strategy as $E_j + 1$ and key recovered as $K_j + 1$.

Step 6. Calculate the P value in eq. 5.1 of S_i, S_j, and $S_j + 1$.

Step 7. If $(S\_j - S\_i) > (S\_j + 1 - S\_i) > \epsilon > 0$, then S_i=S_j, repeat Step 4 $\sim$ Step 6; else if $(S\_j + 1 - S\_i) > (S\_j - S\_i) > \epsilon > 0$, S_i=S_j, repeat Step 4$\sim$ Step 6. Else, choosing the strategy with largest P value among the three strategies S_i, S_j, and S_j+1

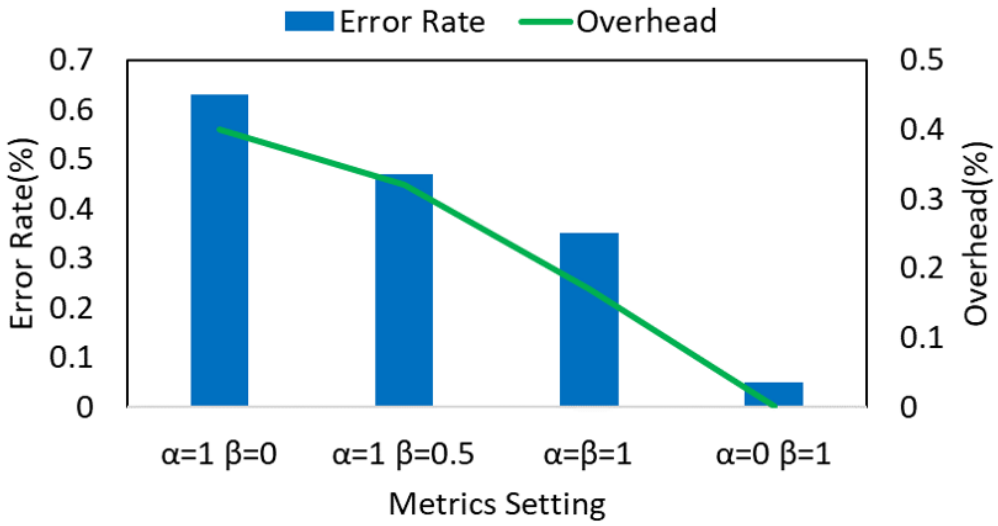


Figure 5.10: Flush+Reload Error Rate and RSA Overhead

5.3.2 Experimental Results

To evaluate the mitigation approach in *Hybrid-Shield*, Flush+Reload and Prime+Probe are used as two case studies. The two attacks belong to two different categories of side-channel attacks: shared-memory and no shared-memory, respectively. As mentioned before, α and β parameters are used to determine the weight of the protection level and performance overhead. We set α, β to various values such as 1, 0; 1, 0.5; 1, 1; 0, 1 as shown in Figure 5.10 and Figure 5.11 and ϵ is set

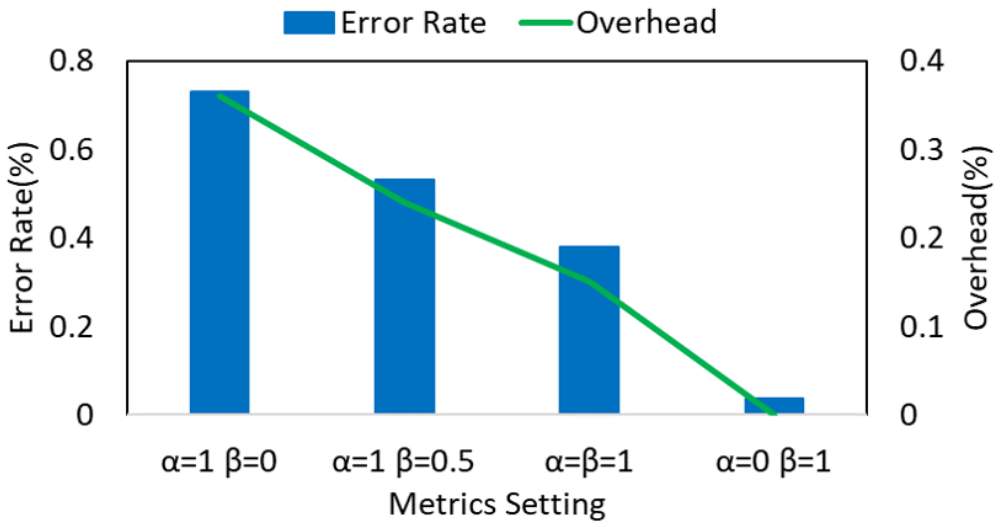


Figure 5.11: Prime+Probe Error Rate and RSA Overhead

as 0.05. One can observe that when α equals to 1 and β equals to 0, the error rate is the highest among the four settings. When α equals 0 and β equals 1, the performance is the most important evaluation metric and no randomization is adopted to protect victim applications. The overhead in both Figure 5.10 and Figure 5.11 is calculated by normalizing the execution time by the time when α equals 0 and β equals 1.

For both Flush+Reload and Prime+Probe, it can be observed that the error rate for attacks can be as high as 52% and 73% when performance overhead is not taken into consideration (α=1, β=0). However, under this mode, the attacks can be significantly undermined. By increasing the weight of the performance overhead, both the error rate and performance overhead decreases. When the error rate and performance overhead are given the same level of importance (α=1 β=1), the SCAs mitigator can result in 35% error rate with 17% performance overhead for Flush+Reload, and 38% error rate with 15% performance overhead for Prime+Probe. Hence, it can be concluded that the proposed SCAs mitigation methodology can effectively pollute victims' cache patterns and change the protection level and performance overhead by the setting.

Latency of Randomization and Adaptation

Randomizing hardware and system level parameters (frequency and prefetchers) requires a time for processors to stabilize prefetchers and instruction decoding units. In this section, we capture the processor's latency for stabilizing prefetchers and instruction decoding components

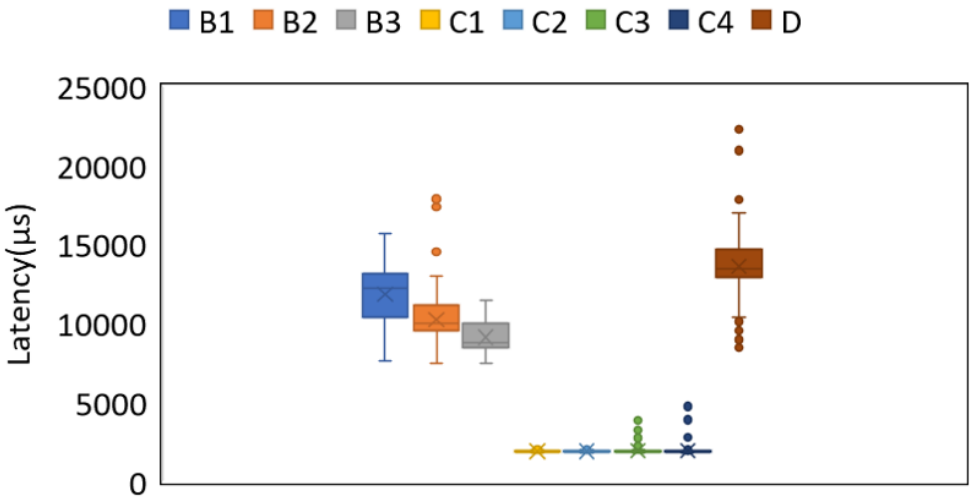


Figure 5.12: Latency of randomizing across A $\sim$ D

when prefetchers or frequency are changing between two values. For instance, in the Scenario B1 the latency corresponds to the time duration between the start of setting CPUs' frequency to 3200MHz and the completion of setting CPUs' frequency to 1600MHz. Since the Scenario A does not require scaling frequency or adapting prefetchers, we collect 100 samples and map them into a box-plot shown in Figure 5.12 across experiment Scenarios B to D. It can be observed that scaling frequency has higher latency compared to adapting the functionalities of prefetchers taking 15 milliseconds and 2 milliseconds, respectively. We believe that scaling frequency results in a more comprehensive influence and requires higher time to achieve the adjustment. Whereas, the prefechers as compared to frequency only influence part of data from memory. Another phenomenon is that scaling frequency with higher range requires more time within 2 to 3 milliseconds difference while adapting various prefetchers results in relatively similar latency, nearly 2 milliseconds. This indicates that more adjustments in processor needs to be done for frequency randomization. Another reason for the latency gap between different frequency scaling ranges is that low frequency takes more time to execute the frequency change.

Performance Overhead Analysis

In this section, we choose SPEC CPU 2006 benchmark suits [153] to evaluate the performance overhead of applications caused by randomizing frequency and prefetchers in terms of application execution time. In order to eliminate the influence of random noise on execution time

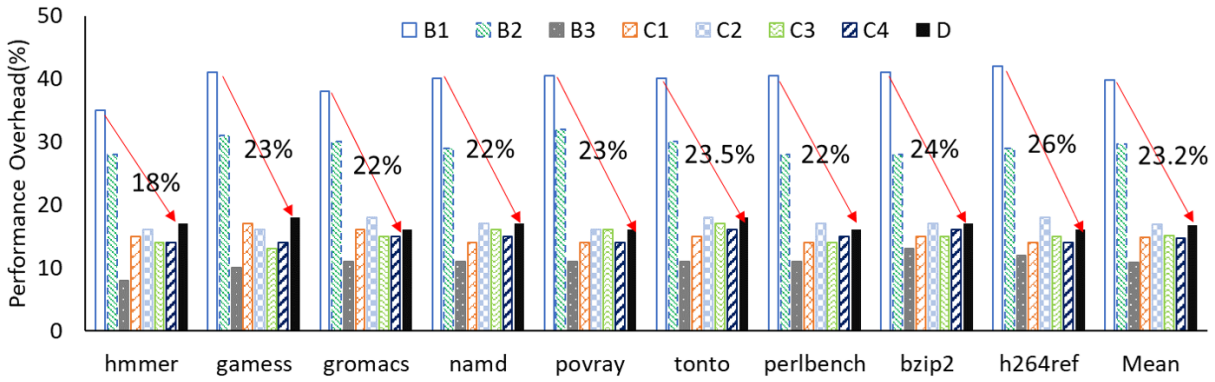


Figure 5.13: Performance overhead analysis of B ∼ D Scenarios across different SPEC CPU2006 benchmarks

caused by system scheduling, etc., each application from the benchmark under the eight scenarios is executed 100 times to avoid the interference of random noises. Figure 5.13 shows the average performance overhead where A to D settings is normalized by the execution time of A.

From Figure 5.13, it can be observed that scaling frequency with largest range (Scenario B1) causes highest performance overhead across all workloads from SPEC CPU 2006 benchmark with average 40% performance overhead. As can be observed from Figure 5.13 by reducing the frequency scaling range (Scenarios B2 and B3), the performance overhead decreases to 28% and 18% highlighting the effectiveness of adapting frequency in lower range for efficient SCA detection and mitigation. The various performance overhead across B1, B2 and B3 is directly caused by frequency. B1 results in applications executed in lowest frequency and yields highest performance overhead. Compared to frequency, disabling various prefechers results in similar performance overhead, ranging from 15% to 17%. Another observation is that the DCU IP prefetcher in Scenario C2 has shown slightly more influential performance reduction to various workloads as compared to three other Scenarios (C1, C3, and C4). As shown, under C2, performance overhead is 17% while the rest three prefetcher adaptation scenarios achieve nearly 15%. On the other hand, Scenario D has shown slightly higher overhead than adapting DCU IP prefetcher (C2) because applications are executed in high (3200MHz) and low frequency (2600MHz) with substantial frequency gap in between. And the error rate of D achieves highest among all other experiment Scenarios. Indicating Scenario D can achieve the most effective protection compared with less performance overhead. Compared to closest state-of-the-art work [54], our proposed randomization methodology in this work based on

frequency scaling and prefetchers adaptation, achieves significantly lower performance overhead, reducing the overhead from 32.66% to around 20% of performance loss. Furthermore, the proposed randomization-based solution can be effectively adopted only when victim applications are executed which are the target of side-channel attacks.

5.4 Conclusion

In this chapter, we analyze the observation change of cache-based side-channel attacks when the underlying system-level and hardware-level settings are changed. In particular, We then thoroughly investigate system (frequency) and architecture (hardware prefetchers) impacts on cache access time to evaluate the potential parameters to adapt against the SCAs. Based on our comprehensive analysis of cache access pattern observations, we propose to randomly scale frequency and adapt hardware prefetchers to effectively mitigate the threats of SCAs. Furthermore, L3 Prime+Probe is deployed as a case study to evaluate the effectiveness of the proposed randomization-based SCA mitigation methodology. The experimental results indicate that under randomization of the frequency with the largest range (Scenario B1), building eviction set shows up to 50% failure rate which results in no sensitive operation sequence of victim application to be observed. The experimental results indicate that under randomization of the frequency with the largest range (Scenario B1), building eviction set shows up to 50% failure rate which results in no sensitive operation sequence of victim application to be observed. Based on the experimental results, we further implement a run-time detection and mitigation of cache-based side-channel attacks to maintain high mitigation effectiveness and incur low performance overhead. For the detection stage, microarchitectural information of victim application under attack and under no attack conditions are collected for training various types of ML classifiers. The ML-based detectors are then deployed to facilitate accurate run-time detection of SCAs using only the HPC features of victim application with high detection accuracy and low instance level false alarm rate. For the mitigation stage, *Hybrid-Shield* offers a lightweight system and architecture level randomization technique to efficiently mitigate the impact of cache-based side-channel attacks with no hardware redesign overhead. By carefully adapting the processor frequency and prefetchers configurations and adding a proper level of noise to the attackers' cache observations, *Hybrid-Shield* protects the critical information from being leaked, enhancing the security of the computer system. The experimental results showed that

the proposed hybrid detection and mitigation countermeasure against timing-based side-channel attacks achieves up to 100% detection accuracy with 0% false alarm rate. After the capturing SCAs, the mitigator component is activated that outperforms the state-of-the-art SCAs mitigation solutions achieving significantly lower performance overhead, reducing the performance loss from 32.66% down to 15%.

Chapter 6

Machine Learning-based Side Channel Vulnerability Analysis for Emerging Applications

As introduced in Chapter 1.3, the research focus of victim applications in side-channel vulnerabilities has been applications of cryptography. Instead, this dissertation aims to reexamine the potential leakage from general applications incurred by side-channel observations. We take web browsers and deep neural networks (DNN) as our target victims and try to build the correlation between side-channel traces and secrets. We prototype a website fingerprinting attack and an inference attack, respectively. We collect the processor's Hardware Performance Counters (HPCs) to build a ML-based classifier for identifying the website users visited. To better evaluate the potential of the fingerprinting attacks, we reduce the sampling granularity to a significantly low level and send the collected samples to a remote attacker's server for conducting classification. Our experimental results indicate that *Leaked-Web* based on a LogitBoost ML classifier using only the top 4 HPC features achieves 91% classification accuracy outperforming the state-of-the-art attacks by nearly 5%. Furthermore, our proposed attack obtains a negligible performance overhead (only <1%) which is around 12% lower than the existing hardware-assisted website fingerprinting attacks. For the inference attack against DNN, the cache-based SCA Flush+Reload is used to collect side-channel traces and label information of input images as the inference attack's target. We explore different settings and classification techniques to achieve a high attack success rate of

stealing label information from the victim models. Additionally, we consider two attacking scenarios: binary attacking identifies specific sensitive labels and others, while multi-class attacking targets recognize all classes victim DNNs provide. Last, we implement the attack on both static DNN models with identical architectures for all inputs and dynamic DNN models with an adaptation of architectures for different inputs to demonstrate the vast existence of the proposed attack, including DenseNet 121, DenseNet 169, VGG 16, VGG 19, MobileNet v1, and MobileNet v2. Our experiment exhibits that MobileNet v1 is the most vulnerable one with 99% and 75.6% attacking success rates for binary and multi-class attacking scenarios, respectively.

6.1 Accurate and Efficient Machine Learning-Based Website Fingerprinting Attack through Hardware Performance Counters

Over the last decades, the Internet has become an essential element of people's social lives to obtain new knowledge/information and conduct businesses and daily tasks. However, new concerns about users' privacy have arisen from the transformation to the virtual world. The users' browsing history could disclose some sensitive information about their background and motifs, such as financial status, sexual orientation, health conditions, or political views. Therefore, by stealing users' online behaviors and access patterns, attackers could further induce personal and sensitive information. This has introduced web browser fingerprinting attacks to violate the users' privacy by extracting and stealing the browsing history of the Internet users. To achieve this, a number of recent works [154, 155] developed attacks that observe device-specific information and website access patterns such as packet sizes, packet timings, and direction of communication to infer the websites that the user is visiting with the aid of ML algorithms.

To protect the online privacy of users, a number of researches have also proposed to hide the network traffic of users [57, 156, 157]. Tor network [57] constructs an overlay network of collaborating servers, called relays. It encrypts the Internet traffic between users and web servers by transmitting the traffic between relays in a way that prevents external observers from identifying the traffic of specific users. The Tor Browser based on the Firefox web browser further protects users by disabling features that may be used for tracking users.Despite considerable progress on developing users' privacy protections mechanisms, there are still a number of recent privacy violation attacks

that rely on the application of ML algorithms that are trained with computer systems' side-channel information collected when a website is open. These attacks stem from existing side-channel vulnerabilities like systems power analysis [158], CPU activity [157], on-chip cache memories [60], memory footprints [58], storage [159], and hardware events [63]. In this work, we comprehensively reviewed the existing studies on web browser fingerprinting attacks and identified some important limitations associated with these methods that could potentially result in an underestimation of the security threats.

- *High Sampling Rate and Performance Overheads:* Existing web privacy violation attacks obtain the data access pattern of the users at 1000-250000 per second sampling rate and employ ML classification techniques to identify the website's characteristics visited by the users with 80%-90% accuracy. Though relatively effective in terms of detection accuracy, such attacks require a high data sampling rate, resulting in large data network traffic and high performance overhead. Therefore, these attacks are easily noticeable by users and the detection systems.

- *Limitation in Monitoring of Websites' Information:* Recent website fingerprint attacks [58, 159] adopt side-channel information such as CPU activity, memory usage, and storage to learn the browsing history of the users. Accurate collection of such information demands high level of isolation in which only a single website should be open on the system. Nevertheless, in real world scenarios a browser could open multiple websites at the same time which has been ignored in existing attack models.

- *The Need to Malicious Website Trigger:* Majority of privacy violation attacks [60, 159, 161] adopt a malicious website to launch the data monitoring process (e.g., the command-line version of Wireshark in [155]). However, visiting the malicious website by the user could lower the threat of the attacks. Moreover, new research study such as [61] has shown that such attacks could be mitigated by combining static and dynamic JavaScript analysis that could successfully detect JavaScript-based attacks.

- *Lack of Analysis on Monitoring and Number of Features:* Our study indicates that prior website fingerprinting attacks have ignored conducting a comprehensive analysis of the impact of different monitoring duration and the number of features collected from websites to infer accurate results. Nonetheless, such analysis is critical to thoroughly evaluate the effectiveness of the deployed attack threats and highlight the importance of adapting better protection mechanisms against such privacy

Table 6.1: Recent Website Fingerprint attacks comparison and contributions of the *Leaked-Web*

Prior Works	**Browser**	**Side-channel Information**	**Attack Model**	**Sampling Rate**	**Duration (s)**	**Performance Overhead**	**Machine Learning**	**Success Rate**
Shane S et.al [160]	Chrome	Power Consumption	Hardware	250,000	15	N/A	SVM	98%
Suman et.al [58]	Chrome, Firefox, Android	App memory footprint	Native Code	100,000	30-40	N/A	Customized Algorithm	N/A
Hyungsub et.al [159]	Chromium Linux, Chrome	Quota Management API	JavaScript	N/A	60	N/A	N/A	90%
Pepe et.al [161]	Chromium Linux, Chrome Mac	Shared event loop	JavaScript	40000	5	N/A	Event Delay Histograms, Dynamic Time Warping	76.7%-91.1%
Anatoly et.al [60]	Firefox, Chrome, Safari, Tor Browser	Cache occupancy	JavaScript	500	30	N/A	Deep Learning	82%
Berk et.al [63]	Firefox, Tor	Hardware Performance Counters	Native Code	25,000	5	N/A	Classic, Deep Learning	86.4%
Sangho [162]	Chromium, Firefox	GPU memory	Native		N/A	N/A	Pixel sequence and histogram matching	95.4%
Qing et.al [163]	Chrome	Power usage	Hardware	10	200k		Random Forest	>90%
Leaked-Web	Firefox	Hardware Performance Counters	Native Code	1	5-60	<1%	Classic, Boosting, Deep Learning, Time-series	80%-91%

violation attacks.

To address the above-discussed limitation, in this work, we thoroughly investigate the security threats brought by exploiting hardware-related features collected from processor's Hardware

Performance Counters (HPCs) and ML classification techniques. The HPCs are special-purpose registers implemented into modern microprocessors to capture the trace of hardware-related events [117]. To this aim, we present *Leaked-Web* a novel, accurate, and efficient ML-based website fingerprint attack model that exploits the information from performance counters with a remarkably small number of samples and performance overheads. Unlike prior works, the presented *Leaked-Web* attack offers the lowest sampling rate, which incurs the least performance overhead to the system. *Leaked-Web* adopts advanced ML algorithms to acquire website browsing history, violating the privacy of the user. Since the HPC events in *Leaked-Web* could be collected from user space with no privileged access and no hardware overhead/modification, which makes the proposed fingerprinting more practical and efficient. To thoroughly analyze the effectiveness of *Leaked-Web*, we also investigate the impact of monitoring granularity and the number of required samples on systems' performance overhead and attack success rate.

In Table 6.1, we have comprehensively analyzed the characteristics of the proposed HPC-based *Leaked-Web* attack as compared with state-of-the-art website fingerprinting attacks across various metrics such as target browser, attack model, performance overhead, the number of samples, deployed ML models, success rate, etc. It can be found that hardware-based, native code-based, and JavaScript-based are three main methods to exploit side-channel information and infer browsing history. However, JavaScript-based attacks [60, 161] can only be launched when the malicious website is visited, greatly undermining the capability of the attacks. Some other works [159] also adopt the native code attack model, while the memory or storage information can only be measured per browser instead of per website. Another drawback of previous works is the high sampling rate which can cause high performance overhead. For example, our analysis shows that monitoring hardware performance counters with a 10,000 sampling rate could incur around 12% performance overhead (will be discussed in detail in Section 3). This work achieves high accuracy with the lowest sampling rate and performance overhead compared to previous work. Furthermore, it does not require visiting malicious websites to launch the attack, which offers higher flexibility and less restriction for *Leaked-Web*. In particular, the main contributions of our work are summarized below:

- We examine the impact of various HPC features on the hardware-based website fingerprinting attack and identify the most prominent low-level features that are crucial to be protected from user space.

- The influence of the number of application traces (sampling rate) and samples per trace are investigated to evaluate the effectiveness and stealing capability of the attack when fewer traces and samples are available.

- We explore the impact of features monitoring on the system's overall performance (e.g., execution time). Hence, our proposed privacy violation attack leads to the least sampling rate, performance overhead, and traffic compared to the state-of-the-art attacks.

- Various ML techniques are comprehensively implemented and evaluated to further demonstrate the effectiveness of the proposed HPC-based website attack, with 91% success rate while obtaining less than 1% performance overhead.

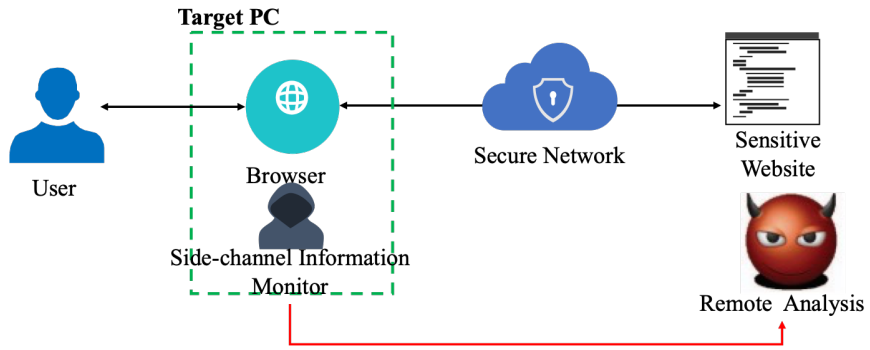
6.1.1 Background

Website Fingerprinting Attacks

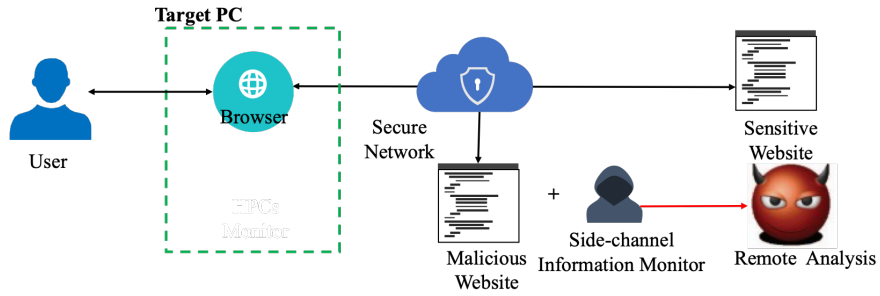
Since website browsing history contains much sensitive information like medical status, political interest, etc., it is critical to prevent leakage, protect users' privacy, and enhance the system's security against potential cyber-attacks. However, prior research has demonstrated that fingerprinting attacks can be independent of operating systems and browsers and rely on side-channel information collected passively. Such attacks can be launched both remotely and locally or via a peripheral device. The side-channel information exists across different computing abstracts, including hardware, system, and network. Once the side-channel information is collected, ML-based classification is leveraged to infer users' visited website information. There are three popular threat models targeting stealing users' browsing history, including native attack model [63], malicious website attack model [60], and hardware attack model [160].

- *Native Attack Model:* In this attack model, the assumption is that the malicious code is resided in the host machine already, which can be done by inserting it into benign applications or downloading accidentally. With this model, attacks can be activated as long as the malicious codes are installed already and do not need users to open certain websites.

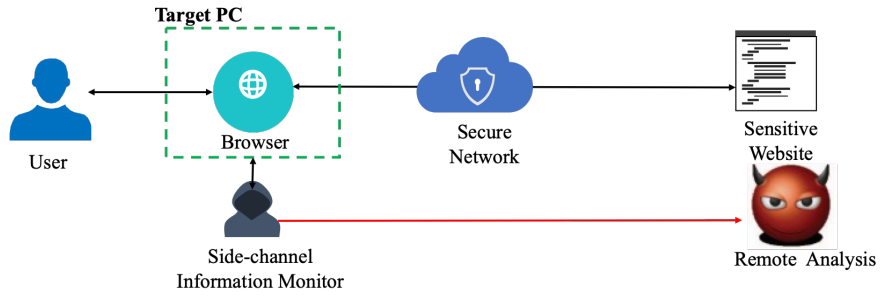
- *Malicious Website Attack Model:* By comparison, this attack model assumes that users click on a malicious website link; thereby, malicious JavaScript codes are executed on local computers. Compared to the native model, this model is more flexible in updating the attack and can be less



(a) Native attack model



(b) Malicious website attack model



(c) Hardware attack model

Figure 6.1: Website fingerprinting attacks threat model

visible since local malware scaning cannot detect malicious codes. The malicious codes are only in charge of side-channel information monitoring for both models without compromising systems.

- *Hardware Attack model:* As shown in Figure 6.1c, this model collects hardware properties, mainly power consumption, when various websites are opened. Some of them infer the side-channel information based on hardware monitoring components such as USB and power meter. Though physical access is needed under this mode, [160] measures the power consumption and achieves 98% website classification accuracy, posing a significant security threat to computer systems and privacy.

Hardware Performance Counters (HPCs)

Hardware performance counters are a set of special-purpose registers built-in modern microprocessors to capture the count of hardware-related events. HPCs have been extensively used to predict the power, performance tuning [127,164,165], attacks detection [18,33,35,85,88,130,166–170], debugging, and energy efficiency of computing systems. It also enhances systems' security by providing microarchitectural information of malware, side-channel attacks, and building detectors based on the events' information [9, 109, 117]. However, recent studies have shown the suitability of such HPCs-based classification for spying on users' behaviors and violating users' privacy [63, 171]. Such privacy breach attacks gain access to HPCs via Perf tool which is a Linux-based low-level performance monitoring tool and provides considerable functionality and abstraction in the kernel, making the interface straightforward for ordinary users [106, 172]. Though Perf is equipped with an access control setting, i.e., $perf\_event\_paranoid$, attacks are still able to access HPCs of applications initiated outside kernel space unless $perf\_event\_paranoid$ equals 4. Hence, the HPCs-based fingerprinting attacks still pose significant threats to system security and users' privacy.

Machine Learning based Classification

- *Boosting:* Boosting aims to enhance the performance of ML algorithms, where the incorrectly classified data from the previous model is employed to implement an ensemble of models. Compared to Adaboost using an exponential loss function, the Logitboost [173] algorithm uses a binomial log-likelihood that changes the loss function linearly. This attribute makes the target model less sensitive to outliers and noise. To the best of our knowledge, no research to date has investigated the performance of the Logitboost algorithm in the field of website fingerprinting attacks. In this work, we applied LogitBoost, as a boosting learning technique on classical ML algorithm RandomForest (RandomF) where the boosted model is abbreviated as Logit-RandomF. - *Deep Learning* Fully Convolutional Neural Network (FCN): A convolutional layer in a deep neural network learns patterns of local structure in the input signal and can learn feature representations over a sequence of input data [174]. FCN is based on the convolutional neural network (CNN) technique, where models employ continuous convolution layers to extract time-series features.

Long Short-Term Memory (LSTM): For the LSTM [175], each temporal trace includes a time-ordered sequence of HPCs on which an LSTM network detects temporal patterns that are

important for discriminating different websites. One and two layers of LSTM neurons with various numbers of neurons per layer were explored. Each node learns a different sequence pattern, and the collection of sequence pattern detectors from all the nodes connected to the output layer are used to classify each HPC temporal sequence.

Time-series Classification: Time-series classification methods deal with such temporal datasets, which represent them in time-domain format and then calculate the distance as the difference between time series [80]. In this work, three prominent classification algorithms are chosen as representatives to compare with the proposed including the dynamic time warping (DTW) [95], Shapelet [77], Bag of Patterns (BOP) [79] are selected as representatives. DTW-KNN determines the best alignment to produce the optimal distance and classifies data according to the calculated distance between time-series sub-sequences. BOP is a structure-based algorithm where a time-series sequence will be transformed into symbolic words while BOP records the frequency of each symbol without order information. Shapelet [77] overcomes the time and space complexity and allows detection for phase-independent shape-based similarity of sub-sequences.

Related Work

Protecting browsing history has emerged recently as a crucial concept to ensure the preservation of users' privacy. In response, [55, 56] are proposed to leverage SSH-based protection methods; [55] encrypts and authenticates messages in one session, and [56] adds cover traffic conservatively while maintaining high levels of security. Similarly, Tor project [57] is one of the most popular traffic transmission approaches, where messages are not directly routed to the receiver but encrypted and forwarded according to ephemeral paths of an overlay network. Though such works have made great progress, there are still some attacks that could bypass such protection mechanisms and extract users' browsing history.

Some recent website fingerprint attacks exploit the clients' machine information when visiting different websites, like memory footprint [58], storage [59], etc. To obtain the information, some attacks prepare a malicious website for users to visit, or a local malware to be launched on the target host. For example, [60] launches a Prime+Probe attack to measure cache occupancy through a malicious website. Then, a deep learning method is leveraged to classify websites and recover users browsing history. Other works such as [58] sample the memory footprint of browsers

through the procfs file system in Linux. To defend against such attacks, [61] proposes an ML-based syntactic-semantic approach that detects browser fingerprinting attacks' behaviors by incorporating both static and dynamic JavaScript analysis. [62] proposes to monitor the running Web objects on user's browser and collect fingerprinting related data. Then, it analyzes them and searches for patterns of fingerprinting attempts. Though effective, they only work for attacks deployed through malicious websites. Furthermore, advanced attacks [63] can be deployed in native code and bypass such detection systems. [64] randomizes properties, like offsetHeight and plugins, to the JavaScript environment, which generates different fingerprints even for the same website and increases non-determinism for attackers. However, randomization is complex and can change a website's visual appearance.

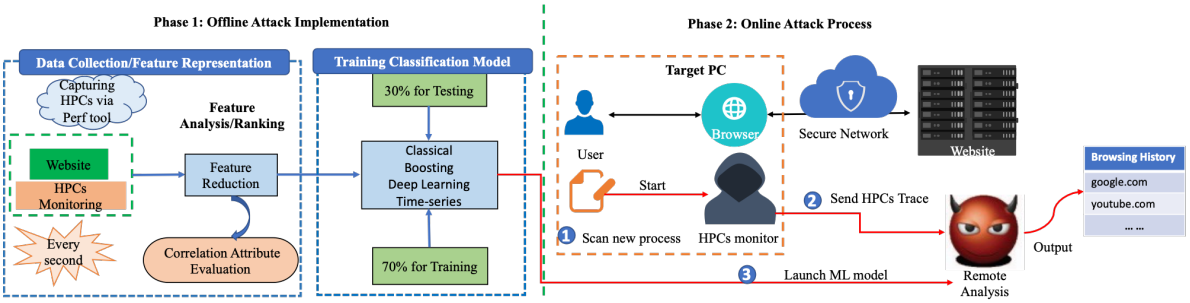


Figure 6.2: Overview of the proposed *Leaked-Web* attack model

6.1.2 Leaked-Web Attack Implementation

This section presents the details of the proposed *Leaked-Web* attack model. *Leaked-Web* is an accurate and efficient HPC-based attack that fingerprints websites with one local HPCs monitoring unit and a remote ML-based trace analyzer, as shown in Figure 6.2. HPCs data are collected for each website during the offline attack implementation phase, and the importance (ranking) of HPC features is evaluated. Next, various ML algorithms are implemented to find the most effective model using a percentage split training-testing method where 70% of data (50 traces per website) is assigned to the training set and 30% of data (20 traces per website) is dedicated to the testing set. Then, the trained ML model is launched and deployed for the online attacking process. For the attacking phase, there are three steps considered in *Leaked-Web*: 1) the browser-related process is scanned every second; 2) HPCs monitoring will be initiated once a new browser process is found; 3) the ML classification model is deployed to predict the website's information based on the

newly collected HPCs trace.

Threat Model

As shown in Figure 6.1a, for our threat model we consider that the malicious codes reside in the host machine, initiate HPCs collection, and send them to a remote attacker. The malicious codes can be launched by inserting it into benign applications or downloading accidentally. With this model, the website fingerprinting attacks can be activated as long as the malicious codes are installed already and do not need users to open certain malicious websites. Furthermore, $perf\_event\_paranoid$ is set less than 4, giving access to reading HPCs from registers. The attacker is able to obtain the potential websites users might open at Alex top site [176].

Experimental Setup

In this work, all experiments are conducted on an Intel I5-3470 desktop with 4 cores and 8GB DRAM, a three-level cache system. In this on-chip cache memory subsystem, while L1 and L2 caches are exclusively separated, the L3 cache memory is inclusive and shared among all cores. In addition, the operating system is Ubuntu 20.0.4 LST with Linux kernel 5.8.0. The proposed HPC-based attack is implemented on a widely used web browser, Firefox.

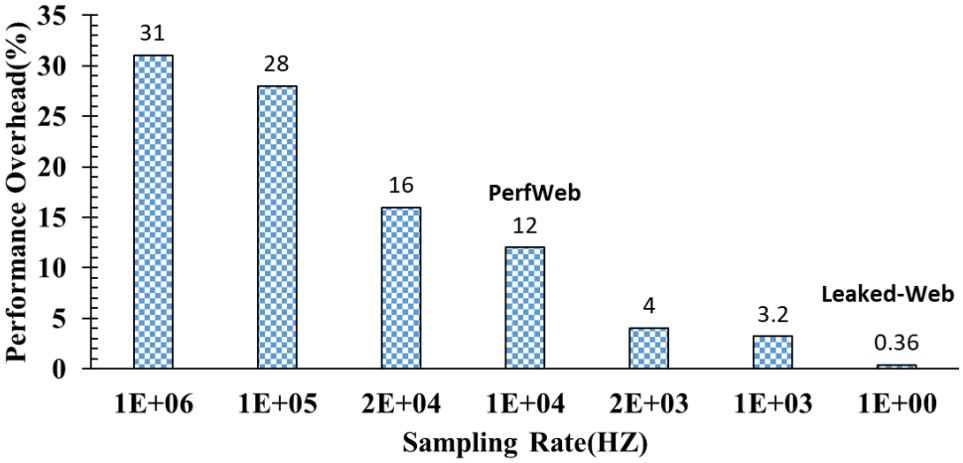


Figure 6.3: Performance overhead with various sampling rates (HZ)

Hardware Performance Events Monitoring

In this work, we use Perf [172] to measure the hardware-related features, and memory and processor's low-level behavior. Perf is a profiling and performance analysis tool that can help to track the hardware performance counters. As introduced in Section 6.2.2, any value less than 4 for */proc/sys/kernel/perf_event_paranoid* gives users access to the HPCs-based profiling of website process. Additionally, we examine the performance overhead and sample size caused by HPCs monitoring in Figure 6.3 in chapter 4. As depicted, the x-axis represents applied the sampling rate ranging from 1^6 to 1^0, the primary y-axis denotes the execution time of victim applications, and the second y-axis represents performance overhead under different sampling rate. Moreover, execution time under no HPCs monitoring is used to obtain the performance overhead percentage. It is observed that generally, the smaller the sampling rate is, the larger the performance overhead is. For instance, when the monitoring scale is 1^6, the performance overhead is at its highest value reaching to 30%. Hence, to make the influence of sampling rate on system performance and the proposed attack less noticeable, we choose 1^0 for the HPCs monitoring in *Leaked-Web*.

Table 6.2: The collected HPC features and their ranking

Rank	**HPC**	**Rank**	**HPC**
1	cache-misses	9	branch-instructions
2	node-loads	10	iTLB-loads
3	branch-misses	11	iTLB-load-misses
4	branch-load-misses	12	dTLB-store-misses
5	LLC-store-misses	13	dTLB-load-misses
6	branch-loads	14	dTLB-stores
7	L1-dcache-stores	15	node-stores
8	L1-icache-load-misses	16	L1-dcache-load-misses

Database Description

We select the top 30 websites from Alexa Top Sites [176]. Similar to previous works no traffic modeling is applied in our database implementation. For the purpose of thorough analysis,

this work considers both Closed World and Open World datasets as described below:

- *Closed World Dataset:* The closed world dataset means that each website is sensitive and exists in training dataset. The proposed attack model considers distinguishing a relatively small list of websites (30) and each websites has 50 traces for training a classification model and 20 traces for testing.

- *Open World Dataset:* Besides the sensitive websites mentioned in the closed world dataset, open world dataset also contains a large set of non-sensitive web pages, all of which the attacker is expected to generally label as "non-sensitive" [60]. For the open world dataset, we add additional 500 traces in which each of them represents the behavior of a single unique website.

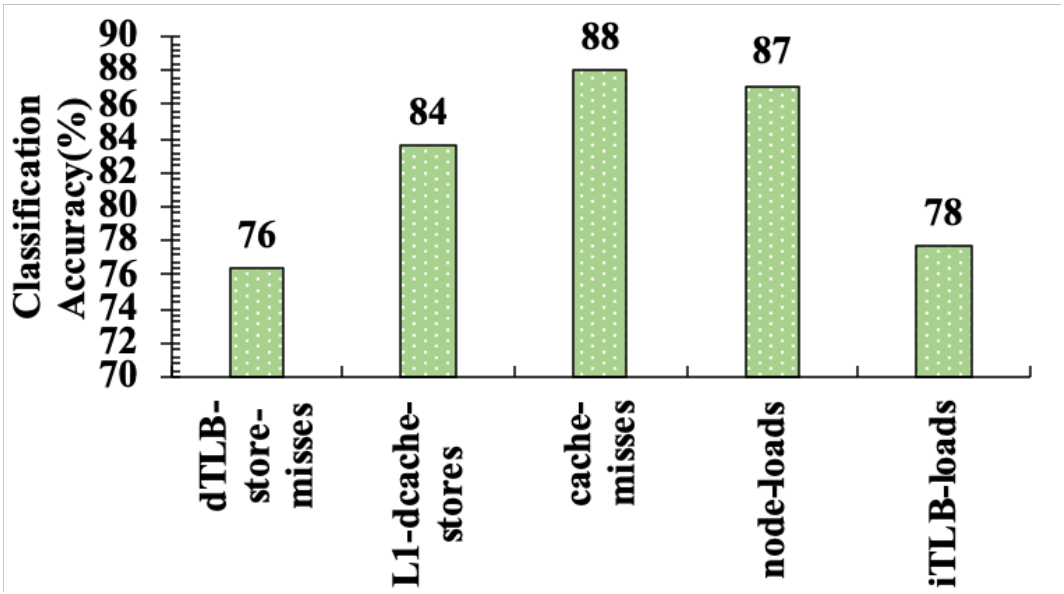


Figure 6.4: The average classification accuracy under Firefox for each HPC

HPC Events Importance Evaluation

Figure 6.4 compares the accuracy of Logit-RandomF-based classifier for websites classification using different HPC events (due to space limitations only 5 events are reported). As can be seen in this figure, changing the HPC could result in over 10% accuracy loss when the same ML classifier is applied. Furthermore, given that there exists a limited number of HPC registers physically available on modern microprocessors' chips that can be accessed simultaneously [120], it is necessary to identify the most important HPCs for classifying the websites. To select the most prominent HPC features we employ Correlation Attribute Evaluation (*CorrelationAttributeEval*

in Weka [177]) with its default settings to calculate the Pearson correlation between attributes (HPC features) and classes (websites). Correlation attribute evaluation algorithm calculates the Pearson correlation coefficient between each attribute and class, as given below:

$$\rho(i) = \frac{cov(Z_i, C)}{\sqrt{var(Z\_i)\ var(C)}} \quad i = 1, ..., 16 \tag{6.1}$$

where ρ is the Pearson correlation coefficient. Z_i is the input dataset of event i ($i = 1, \ldots, 16$). C is the output dataset containing labels, i.e., websites, like "google.com", "youtube.com" and etc. in our case. The $cov(Z_i, C)$ measures the covariance between input data and output data. The $var(Z_i)$ and $var(C)$ measure variance of both input and output datasets, respectively. Next, the HPCs will be ranked according as shown in Table 6.2 and this work chooses the top 4 HPCs for classification.

Machine Learning Classifiers

In the proposed *Leaked-Web* attack, supervised learning is used to model the website fingerprinting attack. The ML implementation stage consists of a building step (training) and attack step (testing). In the building step, multiple traces (50) from each website are collected and labeled. The labeled dataset is used to trainvarious types of ML classifiers, including classical ML, classical ML with boosting, time-series, or deep learning models. The rationale for choosing these machine learning models is that they are from different branches of ML including classical model (RandomForest, LogitBoost RandomForest), deep learning models (FCN, LSTM), time-series models (DTW, BOP, Shapelet) techniques covering a diverse range of learning algorithms that support our comprehensive analysis and experiments. For the attacking phase, the proposed attack model receives unlabeled traces in which each of the trace is corresponding to a user's website visit and the trained classifier outputs the prediction results. Each website has 20 traces for testing and the accuracy is calculated by comparing correctly classified labels and actual samples.

6.1.3 Experimental Results and Analysis

In this section, we comprehensively evaluate the effectiveness of the proposed *Leaked-Web* attack model in terms of classification accuracy and F-measure (F-score) analysis with different ML models, number of HPCs features, monitoring duration. Such analysis gives insight into the cost of HPCs-based fingerprinting attacks and further indicate the requirements of protection approaches.

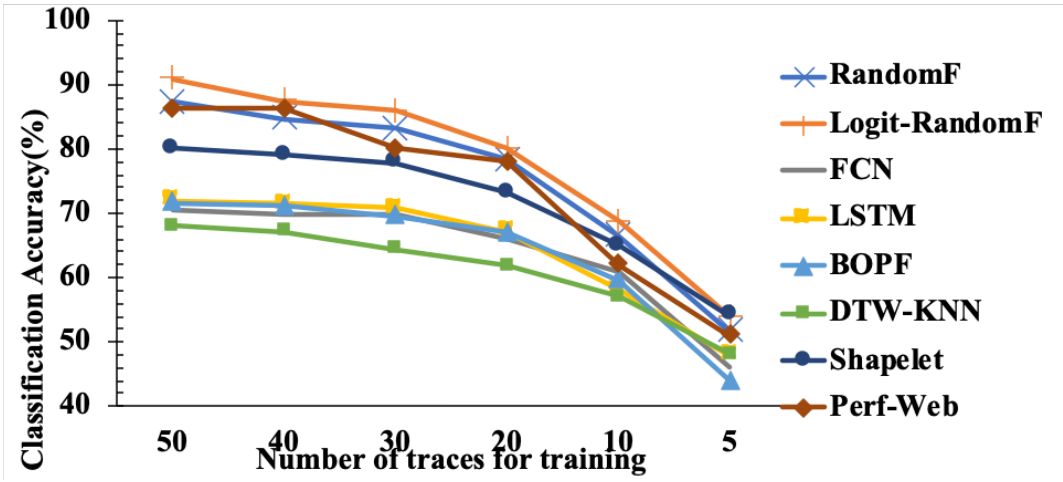


Figure 6.5: Classification accuracy with various classification algorithms with 4 HPCs

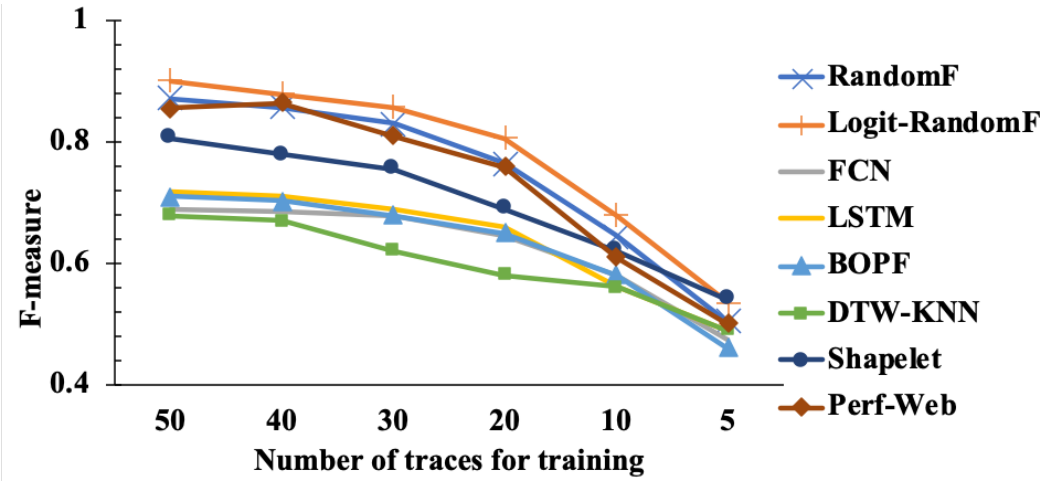


Figure 6.6: F-measure with various classification algorithms 4 HPCs

ML Classification Models Comparison

As introduced in Section 6.1.2, 30 websites selected from the Alexa ranking are executed on our target system and each website has 50 traces for training and 20 traces for testing. Various ML classification models from classic, boosting, time-series and deep learning methods are investigated. As shown in Figure 6.5 and Figure 6.6, X-axis represents the number of traces for training in selected classification models from four ML types (classic, boosting, time-series, and deep learning methods) and Y-axis represents the classification accuracy and F-measure respectively for the closed world. F-measure is interpreted as a weighted average of the precision (p) and recall (r) which is formulated as $\frac{2\times(p\times r)}{p+r}$. The precision is the proportion of the sum of true positives versus the sum

of positive instances, and the recall is the proportion of instances that are predicted positive of all the instances that are positive. F-measure is a comprehensive evaluation metric since it takes both the precision and the recall into consideration. More importantly, F-measure is also resilient to the class imbalance in the dataset, which is the case in our experiments.

As observed from the results, reducing the number of traces in the training phase reduces both classification accuracy and F-measure values. This observation becomes more noticeable as we reduce the traces to lower than 20 where the accuracy becomes below to 80% for all the applied ML classification models. Another observation is that Logit-RandomF classifier outperforms the previous work *Perf-Web* [63] for most of the training sizes (except when the number of traces for training drops to 5). When training traces is 50, Logit-RandomF classifier achieves the highest classification accuracy and F-measure, 91% and 0.901 respectively. As seen, the accuracy is improved in *Leaked-Web* by around 5% from 86% of previous work [63]. Another observation is that the Shapelet-based classification model has shown to be more effective than the other two time-series classification models.

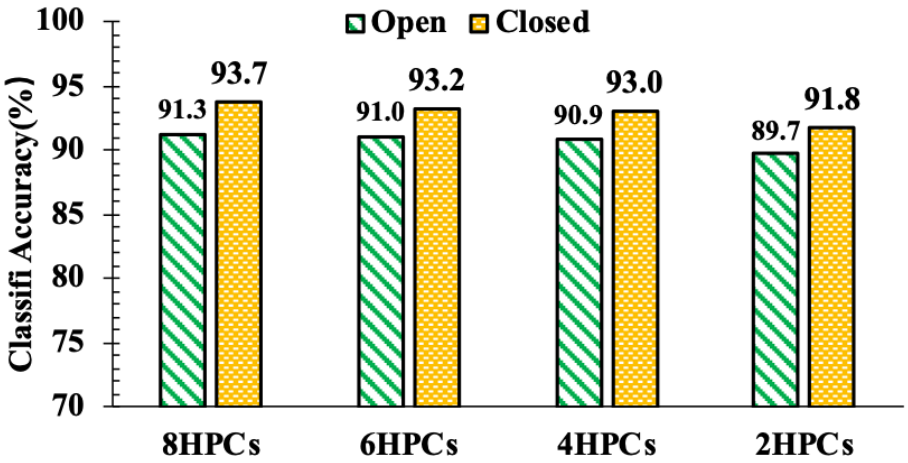


Figure 6.7: Classification accuracy with various number of HPCs features with Logit-RandomF for closed and open world dataset

The number of HPC Features

As our analysis showed that the Logit-RandomF model performs best among all experimented ML models in *Leaked-Web*. Hence, we explore the classification accuracy of Logit-RandomF model under various number of HPCs features and samples, which gives further insights into

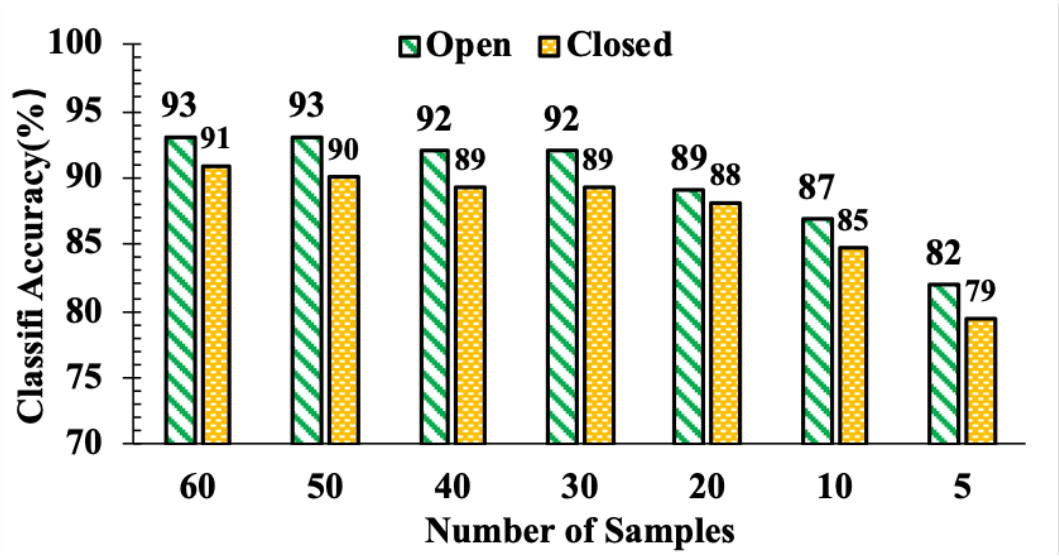


Figure 6.8: Classification accuracy with various number of samples per trace for closed and open world dataset

the potentially vulnerable architectures and duration of attacks. Such analysis is important for evaluating the effectiveness and complexity of future protection mechansims against such privacy violation attacks. This section primarily examines the accuracy of HPC-based website fingerprinting attacks when the number of HPCs changes from 8 to 2, indicating the leakage potential under other architectures with less or more available HPCs registers. As shown in Figure 6.7, the accuracy remains above 89% and 91% for both open and closed dataset when the number of HPCs features reduces from 8 to 2. This indicates that such HPC-based fingerprinting attacks can be effective in accurately inferring users' browsing history at run-time in processor architectures with varying number of HPC registers even with limited available resources (only 2 HPC registers).

Monitoring Duration

In this section, we further investigate the HPCs monitoring duration in order to maintain a high classification performance. Since the number of samples per trace directly decides the duration for data collection when the attack is launched, using less samples indicates that the attack can be applied even when users visit a website within less than 1 minute. As shown in Figure 6.8, X-axis represents the number of samples ranging from 60 to 5, which means monitoring website for 60 seconds to 5 seconds. It can be observed that when reducing the number of samples from 60 to 20, the classification accuracy for closed and open world datasets has slight decrease from 90% to 88%,

and 93% to 89% for closed and open world datasets. However, a further reduction from 20 samples to 10 samples and then to 5 samples per trace causes more significant reduction, from 88% to 84% and then to 79%. Though the noticeable decrease of classification accuracy with only 5 samples, their accuracy still remains around 80%, indicating the capability of inferring the websites of the attacker within 5 seconds.

6.2 Stealthy Inference Attack on DNN via Cache-based Side-Channel Attacks

The advancement of deep neural networks (DNNs) motivates the deployment in various domains, including image classification, disease diagnoses, voice recognition, etc. Since some tasks that DNN undertakes are very sensitive, the label information is confidential and contains a commercial value or critical privacy. This paper demonstrates that DNNs also bring a new security threat, leading to the leakage of label information of input instances for the DNN models. In particular, we leverage the cache-based side-channel attack (SCA), i.e., Flush+Reload on the DNN (victim) models, to observe the execution of computation graphs, and create a database of them for building a classifier that the attacker can use to decide the label information of (unknown) input instances for victim models. Then we deploy the cache-based SCA on the same host machine with victim models and deduce the labels with the attacker's classification model to compromise the privacy and confidentiality of victim models. We explore different settings and classification techniques to achieve a high attack success rate of stealing label information from the victim models. Additionally, we consider two attacking scenarios: binary attacking identifies specific sensitive labels and others while multi-class attacking targets recognize all classes victim DNNs provide. Last, we implement the attack on both static DNN models with identical architectures for all inputs and dynamic DNN models with an adaptation of architectures for different inputs to demonstrate the vast existence of the proposed attack, including DenseNet 121, DenseNet 169, VGG 16, VGG 19, MobileNet v1, and MobileNet v2. Our experiment exhibits that MobileNet v1 is the most vulnerable one with 99% and 75.6% attacking success rates for binary and multi-class attacking scenarios, respectively.

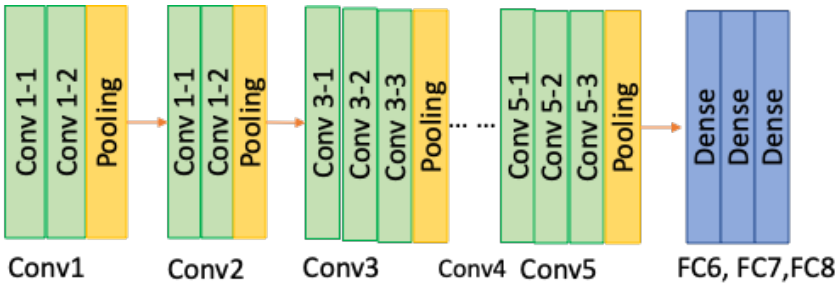
6.2.1 Introduction

Deep neural networks (DNNs) have made significant progress in the past decade and gained increasing popularity in undertaking different tasks, including image classification, language processing, security enhancement, etc. According to the Statista report [178], the global market value for artificial intelligence (AI) is expected to reach 126 billion US dollars by 2025. Similarly, Walls Street Journal [179] issues a report that predicts the advancement in artificial intelligence and machine learning (AI/ML) could fuel the GDP growth by 14% from 2019 to 2030. The advancement of DNNs magnifies the deployment on the edge devices, especially mobile devices with different computation resources and energy restrictions. Integrating deep learning (DL) and mobile apps empowers edge intelligence and provide near real-time insights from data. It is a massive market for companies to attract users and offer higher user satisfaction, bringing a stream of profits. Several successful deep neural network models have been proposed and received notable success, including but not limited to VGG [180], DenseNet [181], etc. The advancement of DNNs magnifies the deployment on both servers and edge devices with different computation resources and energy restrictions [181, 182]. Since then, a number of DNN-enabled applications have been deployed in past decades across various critical domains, such as disease diagnosis [183, 184], intelligent surveillance [185, 186], financial decision [27], and so on.

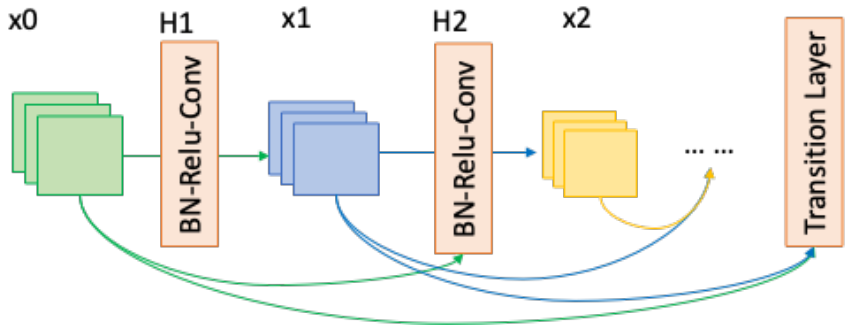
On the other hand, some research finds that DL models routinely leak information about the internal computing process via fluctuating levels of side channels, like cache behaviors. Such leakage can be further used to deduce DL models' architectures or label information. However, the DNN models also bring new security risks — the leakage of label information may cause financial loss and privacy compromise since the label information of such DNN-enabled applications is directly linked to users' crucial decisions and sensitive information. Taking the investment decision-related applications [27] as an example, the leakage of label information can expose the big financial decision to attackers who can take advantage of them and make an illicit profit out of it.

Hence, it is essential to investigate whether DNN models are vulnerable in terms of the leakage of label information. This paper presents a novel inference attack targeting to steal label information of DNN models by leveraging side-channel attacks (SCAs). In particular, we show that a cache-based SCA, e.g., Flush+Reload, can be employed to spy on DNN models' computations, and the extracted observations can deduce the label information of DNN models via ML-based

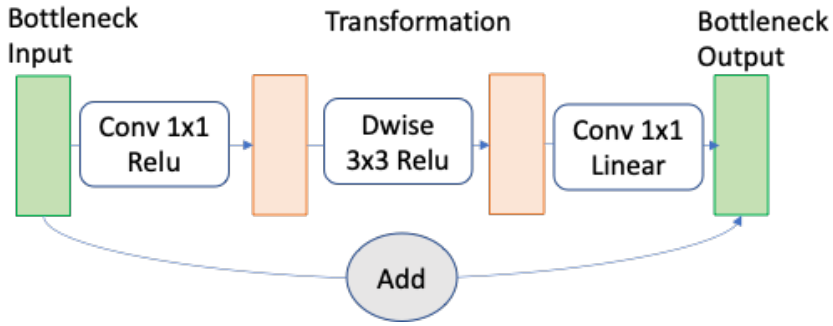
classification techniques. The novelty of our demonstrated attack is to identify a correlation between the cache-based side channel of DNN models' computation trajectory and label information that may contain the users' medical, financial decisions, or other critical information.



a) VGG



b) DenseNet



c) MobileNet v2

Figure 6.9: Architectures of VGG, DenseNet, and MobileNet v2

Though prior works have demonstrated that using side-channel attacks can steal the architectures of DNN models [11, 65, 66, 187], our inference attack is more challenging than reconstructing the architectures of DNN models for the following reasons. a) The black-box DNN models make it almost impossible to infer the computation graphs based on the label information. b) The inference phase takes a much shorter time than loading DNN models, leaving less time for extracting

computation traces for our attack. c) Extracting DNN model architectures uses multiple traces to remove noises while attacking label information demands the attacker obtain label information with only one trace. To address the challenge of noise from SCAs' observations and the incurred accuracy decrease, we explore the various threshold settings for removing repeated DNN models' computation observations and analyze the impact of using different ML techniques for the attacker to decide the optimal one. What is more, we select multiple models with both static neural networks (DenseNet and VGG) and dynamic neural networks (MobileNet) to illustrate the attacker's success rate, i.e., attacker's classification accuracy and highlight the vast existence of the introduced attack. Lastly, this work also identifies the most prominent computations of victim models for the attack, providing insights into the leakage source and future mitigation research. The contributions of our work are categorized as follows:

- To the best of our knowledge, this is the first work to introduce a stealthy attack that steals the label information of DNN models via cache-based SCAs.
- To simulate the attack in the real world, we prototype two variants of the demonstrated attack: a) binary attacking: identifying the sensitive and non-sensitive label information; and b) multi-class attacking: deducing the exact classes of victim models.
- We also identify the most prominent computations of victim DNN models for the attacker to achieve a high success rate, providing insights into the leakage source and future mitigation research.
- We evaluate the attack with two types of victim DNN models, i.e., dynamic neural networks and static neural networks, to manifest the broad existence of the vulnerability.

6.2.2 Background and Motivation

Deep Neural Network (DNN)

DNN models can be categorized into *static* neural networks with the same architectures for all inputs and *dynamic* neural networks with an adaption of architectures and parameters for different inputs. This work selects three DNN families for both static and dynamic DNN models, detailed in the following.

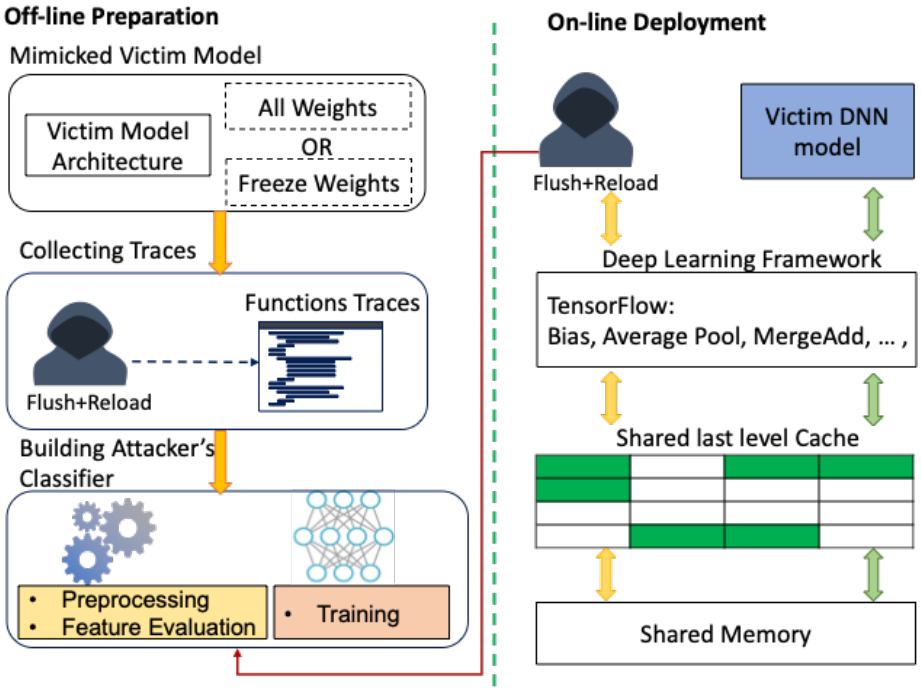


Figure 6.10: Design of the presented attack

- *Static Neural Networks:*

VGG was firstly proposed by Simonyan and Zisserman in 2014 [180] for localization and classification tasks. As shown in Figure 6.9-a), VGG has five blocks of convolutional layers initially and is followed by three fully connected layers. Each convolutional layer has a small kernel size of 3×3 with a stride and padding to maintain the same spatial dimensions as the last layer. The depth of VGG varies from 16 to 19 layers, and each of them is a variant DNN model: VGG 16 and VGG 19.

Dense Convolutional Network (DenseNet) as shown in Figure 6.9-b), employs a feed-forward fashion to connect each layer to every other layer in the network. It maximizes the information flow to alleviate the vanishing gradient problem and strengthen feature propagation [181]. Specifically, for lth layer, there are l inputs from all preceding convolutional blocks, and its output sends to the $L - l$ subsequent layers. This work chooses two variants of the DenseNet family: DenseNet 121 and DenseNet 169.

- *Dynamic Neural Networks:*

MobileNet v1 [182] and MobileNet v2 [188] were designed for conducting classification, detection, and other computer vision-related tasks in mobile devices. They enable applications

installed in mobile devices to equip more functionalities with human and real-world interaction based on deep learning neural networks. MobileNet v1 [182] mainly leverages depthwise separable filters, width multiplier, and resolution multiplier to balance the accuracy loss and computation size. As depicted in Figure 6.9-c), MobileNet v2 includes another two techniques, a) linear bottlenecks between layers, and b) shortcut connections between the bottlenecks.

Cache-based Side-Channel Attacks

To bridge the latency between memory and CPU, cache hierarchies are introduced and shared among applications. The shared cache gives the attacker opportunity to influence applications' memory access and infer their cache access pattern by measuring its accessing latency, termed as cache-based side-channel attacks (SCAs). The existing cache-based SCAs, including [1, 2, 18, 167], spy on shared cache activities and steal critical information, including passwords, secret keys, etc. In this work, we leverage *Flush+Reload* to observe the computation behaviors of victim DNN models. This attack targets the *Last-Level Cache* in the CPU, flushes out victim applications' data in the cache and waits for the victim application to execute. After flushing the cache, the attacker tries to access the data and measures the accessing time (latency). A shorter accessing time denotes that the victim application has accessed the data; otherwise, it has not been accessed.

Motivation of the Attack

Deep neural networks (DNNs) have made significant progress in the past decade and gained increasing popularity in undertaking different tasks, including image classification [180, 189, 190], language processing [191, 192], security enhancement [193], etc. Several successful deep neural network models have been proposed and received notable success, including but not limited to VGG [180], DenseNet [181], etc. The advancement of DNNs magnifies the deployment on both servers and edge devices with different computation resources and energy restrictions [181, 182]. Since then, a number of DNN-enabled applications have been deployed in past decades across various critical domains, such as disease diagnosis [183, 184], intelligent surveillance [185, 186], financial decision [27], and so on. Their label information either contains critical information or impacts significant decisions, which attackers can steal and make an undesired profit out of them or conduct crimes based on them. In the finance domain, attackers can misuse the investment suggestions stolen

from victim DNNs. Energy controlling systems [28] leverage DNNs to design the optimal online power control policy while the attacker with label information can take advantage of the policy information to deliberately overload the energy network and cause a denial-of-service attack on customers. Hence, we believe that the leakage of such label information of sensitive DNNs-enabled applications can allow attackers to cause undesirable damages.

6.2.3 Overview of Attack

This work demonstrates a novel attack that stealthily spies side-channel information to deduce labels of inputs. As depicted in Figure 6.10, the whole attack contains two parts: *offline preparation* and *online deployment*. Offline preparation firstly mimics the victim model based on architecture and weights knowledge. And then, it collects Flush+Reload traces during the inference phase of the mimicked model to build the attacker's classifier. Once trained and tested, Flush+Reload is launched online to collect the traces of the victim model and send them to the attacker's classifier for deducing the label information of inputs. We introduce the details of the attack in the following sections.

Threat Model

The attacker's goal is to secretly infer the labels of instances sent to the victim DNN models. We assume that the attacker (without sudo access) resides in the same physical machine with DNN models (also referred to as victims) and can launch a software program on the machine. The victim is a DNN model and takes images as input, the output of which is labels and needs protection. We assume that the attacker *does not have direct access* to the *input images* and *label information*.

- *Assumption:* The attack aims to obtain the label information of instances sent to victim DNN models and has the knowledge of victim models' weights or untrainable weights. We also assume that the attacker knows the label candidates of victim models and the type of dataset victim models are targeting, e.g., flowers classification [194]. As for the knowledge of victim models' architectures and parameters, we have two assumptions: setting A and B, as listed below.

- Setting A: the attacker has the full knowledge of the victim model, including architectures, weights, and parameters, which can be acquired based on approaches studied in prior research

[66], or victim models use public weights.

- Setting B: the attacker has the knowledge of the victim model and the pre-trained weights used while not knowing the newly added layers at the end of the model. Additionally, victim models freeze pre-trained parameters and weights and only train the newly added layers.

- *Target of the Attack:* We consider two attacking scenarios as detailed in the following.

- *Binary Attacking:* the attacker targets stealing one particular class of labels and categorizes the rest classes of labels as "others." The binary classification accuracy can directly reflect the ability to steal the labels from victim DNN models with a specific value that attackers are interested in.

- *Multi-class Attacking:* the attacker tries to identify all classes of labels just as victim models do.

Table 6.3: Monitored function list

Bias	Sigmoid	RunHelper	Average Pool
Relu6	Depthwiseconvop	End Conv	Max Pool
MatMul	Depthwiseconv2d	Tanh	LaunchConv
Elementwise	Mulop	Elu	Concat
Merge Add	Selu	Softsign	Softplus

Collecting Training Traces

To observe the computations of victim DNN models, we select functions listed in Table 6.3 and each function corresponds to the specific architecture of victim DNN models. Though there are 20 functions selected during the offline preparation, only a subset of the functions is chosen for the online deployment. Our attacker and victims run at the user level on the same operating system and the host machine. The attacker runs the mimicked DNN model with a similar database as the victim while Flush+Reload is initiated to monitor the target functions simultaneously.

Building Attacker's Inference Model

- *Prepossessing:* Since traces obtained via Flush+Reload are noisy due to transient execution, we need to take a further step to clean traces and filter out some functions. Since Flush+Reload might suffer from repeated observations when the computations of DNNs continue, we employ the number of cycles between two computations to clean noises in traces, referred to as "threshold" in the following sections. In Section 6.2.4, we demonstrate the influence of threshold on the classification accuracy of the attacker's inference model. After removing noises, all observations are converted into the occurrences of each computation.

Table 6.4: Prominent functions for the introduced attacker

Mobilenetv1	Mobilenetv2	DensetNet121	DensetNet169	VGG16	VGG19
Bias	Bias	Maxpool	Mulop	Bias	Mulop
Mergeadd	Mergeadd	LaunchConv	Mergeadd	Mulop	Mergeadd
LaunchConv	LaunchConv	Concat	LaunchConv	Maxpool	LaunchConv

- *Feature Evaluation:* Though the introduced attack has access to monitor several functions as listed in Table 6.3, we conduct an importance evaluation for each function for two reasons: a) to minimize the number of monitored functions to incur the least the influence of our attack on victims and reduce the possibility of being detected; and b) the importance of functions reveals the leakage source and can provide more insights for future mitigation works. To achieve this, we leverage the correlation-based feature selection (CFS) with the greedy-stepwise search algorithm [84] approach to select the optimal subset functions for implementing an attack with a high success rate, i.e., high classification accuracy. After the importance evaluation, we observe that *Bias, Merge Add, Launch Conv* are commonly effective for attacking all six victim models. Hence, only the most prominent functions are monitored for online deployment.

- *Training Classifier:* We consider two types of attacking purposes: identifying the sensitive inference model (binary attacking) and identifying the precise class of victim's inputs (multi-class attacking). To build a robust classifier with higher accuracy, we split the whole dataset into training and validation in case of an over-fitting issue. Furthermore, we consider a vast range of ML classifiers in this work to select the optimal one with the highest accuracy. Five classifiers are selected: OneR, J48, SVM-based SMO, KNN, and Multi-Layer Perceptron (MLP). The rationale

for selecting these ML models is that they are from different branches of ML, including rule-based, tree-based, support vector machine, lazy learning-based, and neural network techniques covering various learning algorithms.

Online Deployment

As presented in Figure 6.10, we deploy the Flush+Reload on the same host machine as victim models and send the observations to the remote attacker's classifier to extrapolate the labels of victim models' inputs. The detailed steps are listed below:

- **Step 1:** Launch Flush+Reload on the same host machine as victim models and collect traces of function calls once the victim model is started.
- **Step 2:** Preprocess the data collected online with the same preprocessing approach (6.2.3) as the one in preparation.
- **Step 3:** Send processed data to the attacker's classifier for deducing the inputs' label information of the victim model. The deduced label information can be further leveraged by other malicious activities, like stealing investment decisions, as discussed in Section 6.2.2.

6.2.4 Evaluation

Experiment Setup

- *Attack Platform* All evaluations are done on a Dell server with 32 cores Intel(R) Xeon(R) CPU E5-2683 v4 and a three-level cache system. L3 cache memory is inclusive and shared among all cores meaning that flushing out the data in the last-level cache could remove the data in the L1 cache. The inclusiveness of the L3 cache creates a potential vulnerability surface for last-level cache attacks to be exploited. Our proposed inference attack should be effective on other platforms which are also vulnerable to Flush+Reload platforms.

- *Victim Models for Evaluation* As discussed in Section 6.2.2, we select three prevalent deep learning model families, each with two variants as victims: DenseNet 121, DenseNet 169, VGG 16, VGG 19, MobileNet v1, and MobileNet v2 with implementation in Tensorflow 1.10.0 and Keras with the default weights, "ImageNet."

- *Datasets* Flowers dataset [194] is used in this work for evaluating the success rate of the proposed attack. Since we consider two attacking scenarios as detailed in Section 6.2.3 which demand no training and retraining, respectively, we split the dataset with two approaches. For setting A (no training), all data are split into 10%-90% to build the attacker's classifier and evaluate the attack online. For setting B (retraining), we split the dataset into 50%-10%-40% for retraining the last layer of the victim models, building the attacker's classifier, and testing the proposed attack.

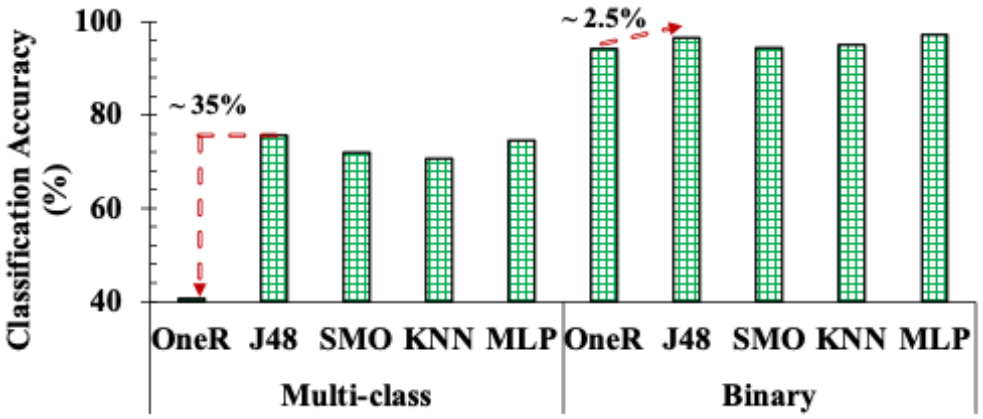


Figure 6.11: Classification accuracy for binary and multi-class with various classifiers with the attack on MobileNet v1

Table 6.5: Binary attacking success rate: sensitive labels (daisy) vs others (dandelion, roses, sunflowers, and tulips)

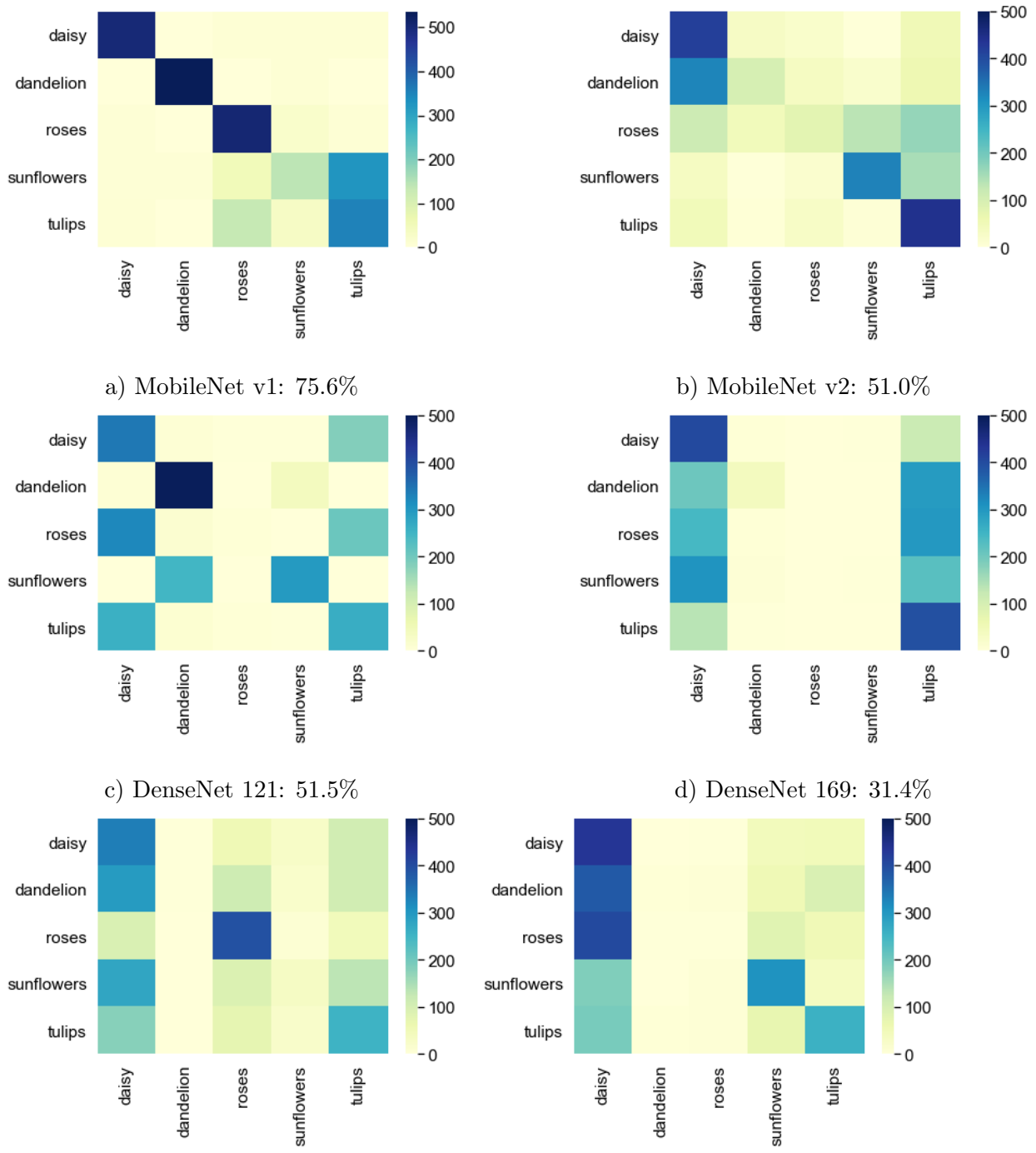
Settings	Threshold (Cycles)	MobileNet v1	MobileNet v2	DenseNet 121	DenseNet 169	VGG16	VGG 19
Setting A	0	94.3	69.4	73.9	44.9	58.8	56.9
	100	**99.0**	**77.3**	73.7	52.4	60.5	65.5
	200	98.8	66.3	73.6	51.6	60.5	68.7
	500	96.1	65.6	**78.5**	**66.6**	**70.6**	**69.9**
Setting B	0	93.9	69.0	73.0	52.2	58.5	56.2
	100	98.3	**76.9**	73.6	44.9	60.4	65.3
	200	**98.4**	66.1	73.0	50.7	60.3	69.4
	500	95.5	65.4	**78.3**	**66.4**	**69.7**	**69.8**

Attacking Success Rate with Various Classifiers

We explore different classification algorithms to select the optimal one for deducing label information from the victim model based on Flush+Reload observations. As shown in Figure 6.11, we present the binary and multi-class attacking success rate, i.e., classification accuracy, of our attack on MobileNet v1. Generally, multi-class attacking is more complicated than the binary attacking scenario with around a 20% accuracy difference. Selecting an appropriate classifier for multi-class attacking is more critical for our attack since OneR yields 35% less accuracy than J48 with 75% multi-class classification accuracy. By comparison, classification accuracy is less critical for binary attacking scenarios, while J48 is better than OneR by around 2.5%. For both binary and multi-class attacking scenarios, J48 and MLP outperform the rest three classifiers.

Binary Attacking Evaluation

As seen in Table 6.5, we consider the classification accuracy with six different victim models under noise removal thresholds ranging from 0 to 500 cycles under both setting A and setting B detailed in Section 6.2.3. We observe that similar results are found for setting A and setting B, indicating that introducing customized layers with pre-trained weights freeze does not cause a noticeable impact on our attack. For all six victim models, we observe a noticeable influence of changing noise removal threshold on classification accuracy under setting A and B while the impact varies from models. We can observe that the attacking success rate on both MobileNet v1 and MobileNet v2 receives the highest value when the threshold is set at 100 cycles. By comparison, the static neural network models, DenseNet and VGG, demand a higher threshold value at 500 cycles. The difference is that the filter shape of MobileNet convolutions is generally smaller than VGG and DenseNet, having less execution time. Across all six models from three DNN families, MobileNet v1 is the most vulnerable one and suffers over 99% attacking success rate when the noise removal threshold is set at 100 cycles. Though DenseNet and VGG are less vulnerable, we still observe 78.5% and 70.6% success rates under setting A and similar results under setting B, indicating the attacker can still devise the label information from the victim static neural network models. Another observation is that static models from the same DNN family, i.e., DenseNet and VGG, experience similar attacking success rates with various threshold cycles. Though MobileNet v2 are from the same DNN family as MobileNet v1, the attacking success rate is 77.3% because



a) MobileNet v1: 75.6%

b) MobileNet v2: 51.0%

c) DenseNet 121: 51.5%

d) DenseNet 169: 31.4%

e) VGG 16: 37.7%

f) VGG 19: 37.4%

Figure 6.12: Heatmap of multi-class attacking success rate under setting A of six models: MobileNet v1, MobileNet v2, DenseNet 121, DenseNet 169, VGG 19, and VGG 16.

its short cuts between bottlenecks and depthwise causes the occurrences of *LaunchConv* function challenging for attackers' classifier. In conclusion, binary attacking can receive up to around 99%

success rate and yield a significant difference in success rate on the six victim DNN models.

Multi-class Attacking Evaluation

Besides the binary classification that helps our attacker identify sensitive labels from victim models' execution, we also evaluate the attack to steal all possible labels individually from victim models. We plot the predicted labels from our attacker's classification results and ground truth labels with heatmaps as shown in Figure 6.12 for setting A with accuracy included. Similar to binary classification, our attack yields similar accuracy under the two settings, while results for setting B are not presented for the brevity of space. Compared to the binary attacking, inferring the multiple labels, i.e., five in this work, is more challenging for the attack, suffering over 20% attacking success rate decrease for almost all victim models. Still, the multi-class attacking on MobileNet v1 also receives a 75.6% success rate. Most classification errors are from sunflowers and tulips, meaning the attack receives high confidence for the other three classes samples. Similar to the binary attacking scenario, MobileNet v2 is less vulnerable since the occurrences of *LaunchConv* function are less separable caused by shortcuts and depthwise. Still, we find that most samples from *daisy*, sunflowers, and tulips are classified correctly by the attacker's classifier. By comparison, the four static DNN models, i.e., DenseNet 121, DenseNet 169, VGG 16, and VGG 19, are less likely to suffer from multi-class attacking.

6.3 Conclusion

This chapter examined the side-channel vulnerability on two emerging applications, i.e., web browsers and deep neural network models, to further highlight the wide existence of side-channel vulnerability in computer systems. To automate the process, this chapter uses ML classification approaches to analyze the correlation between side-channel observations and sensitive information. For the web browser application, we investigated the website fingerprinting attacks which take advantage of the microarchitectural state of the processor running the browser. While Hardware Performance Counters (HPCs) are widely used for performance tuning, application profiling, malware detection, etc., this work presents *Leaked-Web*, a fast and unified HPC-based attack model that collects microarchitectural HPC samples by using the Perf tool under Linux and trains accurate and efficient ML classifiers with the HPCs' traces. Compared to prior works, *Leaked-Web* demands

significantly lower network traffic per website visit and achieves up to 91% classification accuracy outperforming the state-of-the-art attack by nearly 5%. Our presented attack obtains a trivial performance overhead (less than 1%) which is more than 12% lower than the existing HPC-based attacks. Then we explored the potential label leakage with cache-based SCA Flush+Reload on deep neural networks. To achieve it, we build an effective classifier that predicts the label information with Flush+Reload traces by investigating different noise removal settings and exploring a broad range of classification techniques. Additionally, we illustrate both binary attacking and multi-class attacking capability with six DNN models from both static and dynamic neural networks. Our experiments exhibit that the attack achieves up to 99% binary attacking success rate and 75.6% multi-class attacking success rate on MobileNet v1.

Chapter 7

Future Work and Conclusion

An increasing number of side-channel vulnerabilities have been identified in the past decade, which notably compromise the security of computer systems. Hence, it is urgent to countermeasure such side-channel attacks and design effective defense approaches. In this dissertation, we propose the use of ML for developing effective SCAs detectors with hardware-level features, implementing a light-weight mitigation framework with a minor performance overhead, and analyzing the leakage potential in emerging applications.

In Chapter 3, we introduced the two customized ML-based SCAs with hardware performance counters to address the challenges in real-time SCAs detection and zero-day SCAs capturing, respectively.

To cope with the increasing number of edge devices in the following generation networks, Chapter 4 also explored the potential of applying ML to build effective attack detectors against malware and SCAs on edge devices including autonomous vehicles, mobile, and laptops.

Chapter 5 investigates the effectiveness of randomizing system-level and hardware-level settings, i.e., frequency and prefetchers, to pollute the observations of SCAs. Based on the experimental results, adjusting the frequency of processors and adapting prefetchers can effectively help obfuscate the executions of victim applications. We further proposed *Hybrid-Shield* which combines the ML-based SCAs detector proposed in Chapter 3 and the randomization-based mitigation approach to achieve a low performance overhead with high protection level.

Besides applications of cryptography like RSA, AES, Chapter 6 probed the side-channel vulnerabilities in emerging applications, including the web browser and deep neural network models.

We adopted ML classification techniques to analyze the side-channel traces and expose the correlation between side-channel observations and sensitive information, i.e., website and label information.

Though this dissertation addressed the security challenges from SCAs, there are still a lot of open research problems. Following, we will present potential future works firstly and then conclude our work in this dissertation.

7.1 Future Directions

The emerging, more sophisticated side-channel attacks impact a diverse range of computing platforms, from edge to cloud continuum, threatening computer systems owned by individuals, organizations, or governments. Compared to software-oriented malware, hardware-oriented SCAs demand more reliable detection and defense approaches beyond upgrading since they exploit hardware vulnerabilities. The advancement of next generation networks and ML fosters a more connected and autonomous world where such malicious activities can result in breaches of privacy, malfunctions of critical facilities, and even loss of lives. However, the security of edge ML routinely leaks information about the internal computing process via fluctuating levels of power consumption, electromagnetic emissions (EM), and cache behaviors.

In future research, one direction is to assess the potential side-channel leakages of different ML models deployed on edge devices like mobiles or other wearable devices. Existing works use effectiveness and energy efficiency as two primary metrics used for deciding the optimal ML model. Future research can include the security in terms of side-channel vulnerabilities as a third metric to benchmark the leakage difference across various ML models. To secure the edge devices against SCAs, future research is required to build a secure, tamper-proof foundation. The defense can include two aspects: detection and mitigation. For the detection aspect, future research needs an ML-based on-device detection to equip edge devices with awareness of SCAs. The second aspect is to design a mitigation framework with zero side-channel emissions and an automated validation platform to evaluate the effectiveness of our countermeasures in reducing side-channel leakages in ML edge devices.

Besides the side-channel vulnerability evaluation of ML models, another open problem is the security of heterogeneous architectures like CPU-FPGA, CPU-GPU, etc. The design of the heterogeneous computation platforms significantly improves computation speed with less energy

cost, especially for CPU- intensive workloads, like DNN training and inference. To meet the demand, the industry, including Google, Intel, and Amazon, also includes integrated products of CPU and customized hardware accelerators in their cloud services. However, isolation and security boundaries have mainly been studied and implemented in the CPU, while the heterogeneous architectures introduce new attack vectors and vulnerabilities. Moreover, cloud providers fasten the deployment of accelerators-equipped computation platforms, leading the heterogeneous architectures shared among multi-tenants to threaten data security. In the future, it is crucial to reexamine the isolation and security boundary in heterogeneous architectures and develop an automation tool to find the potential side channels and secure the heterogeneous architectures via obfuscation at system and application levels.

7.2 Conclusion

This dissertation presented the security threats brought by emerging attacks, i.e., side-channel attacks (SCAs), which can notably compromise a computer system's privacy. We demonstrated that ML techniques could be extensively used in side-channel evaluation and defenses to address the security challenges. We explored the feasibility and effectiveness of leveraging ML techniques and hardware-level features to design SCAs detectors. To further defend against SCAs, we also investigated a light-weight mitigation approach by randomizing system-level and hardware-level settings, including frequency and prefetchers. Besides the analysis of SCAs on cryptography applications, this dissertation also examined the leakage potential of general applications and found that both web browser and deep neural networks are susceptible to side-channel attacks and are prone to losing privacy assurance. Lastly, we discussed the open research problems that are left behind in this dissertation. The progress of the next generation network and high performance hardware exacerbates the deployment of ML and heterogeneous architectures while they bring new attacking surfaces. The privacy assurance of ML in terms of side channels needs to be further re-evaluated. Moreover, the heterogeneous architectures, including CPU-GPU, and CPU-FPGA, break traditional isolation and security boundary, which can also bring new side channels and incur sensitive information disclosure. We leave these research problems for future investigations.

Bibliography

[1] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on S&P*, 2015.

[2] Yuval Yarom and Katrina Falkner. Flush+ reload: A high resolution, low noise, l3 cache side-channel attack. In *USENIX Security Symposium*, volume 1, pages 22–25, 2014.

[3] Daniel Gruss and et al. Flush+ flush: a fast and stealthy cache attack. In *DIMVA*, pages 279–299. Springer, 2016.

[4] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. Spectre attack: https://github.com/eugnis/spectre-attack.git.

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

[6] Marina Minkin, Daniel Moghimi, Moritz Lipp, Michael Schwarz, Jo Van Bulck, Daniel Genkin, Daniel Gruss, Frank Piessens, Berk Sunar, and Yuval Yarom. Fallout: Reading kernel writes from user space. *arXiv preprint arXiv:1905.12701*, 2019.

[7] Stephan Van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Ridl: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 88–105. IEEE, 2019.

[8] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 753–768, 2019.

[9] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Cloudradar: A real-time side-channel attack detection system in clouds. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 118–140. Springer, 2016.

[10] Mulong Luo, Andrew C Myers, and G Edward Suh. Stealthy tracking of autonomous vehicles with cache side channels. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 859–876, 2020.

[11] Sanghyun Hong and et al. Security analysis of deep neural networks operating in the presence of cache side-channel attacks. 2018.

[12] Sanghyun Hong, Michael Davinroy, Yiğitcan Kaya, Dana Dachman-Soled, and Tudor Dumitraş. How to 0wn nas in your spare time. *arXiv preprint arXiv:2002.06776*, 2020.

[13] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406, 2018.

[14] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles K Nicholas. Malware detection by eating a whole exe. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

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

[17] Maria Mushtaq, Ayaz Akram, Muhammad Khurram Bhatti, Maham Chaudhry, Vianney Lapotre, and Guy 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*, pages 1–8, 2018.

[18] Han Wang, Hossein Sayadi, Setareh Rafatirad, Avesta Sasan, and Houman 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)*, pages 1–6. IEEE, 2020.

[19] Samira Briongos, Gorka Irazoqui, Pedro Malagón, and Thomas Eisenbarth. Cacheshield: Detecting cache attacks through self-observation. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, pages 224–235, 2018.

[20] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers' Track at the RSA Conference*, pages 1–20. Springer, 2006.

[22] Fangfei Liu and et. al. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *2016 HPCA*.

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

[24] D Page. Partitioned cache architecture as a ėide-channel defence mechanism. 2005.

[25] Zhenghong Wang and Ruby B Lee. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*.

[26] Fangfei Liu and Ruby B Lee. Random fill cache architecture. In *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[27] Gyeeun Jeong and Ha Young Kim. Improving financial trading decisions using deep q-learning: Predicting the number of shares, action strategies, and transfer learning. *Expert Systems with Applications*, 117:125–138, 2019.

[28] Mohit K Sharma, Alessio Zappone, Mérouane Debbah, and Mohamad Assaad. Deep learning based online power control for large energy harvesting networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8429–8433. IEEE, 2019.

[29] TIMOTHYB.LEE. Tesla's autonomy event:impressive progress with an unrealistic timeline. 2019.

[30] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 17–32, 2014.

[31] Han Wang, Hossein Sayadi, Setareh Rafatirad, Avesta Sasan, and Houman Homayoun. Scarf: Detecting side-channel attacks at real-time using low-level hardware features. In *IOLTS*. IEEE, 2020.

[32] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Hybridg: Hybrid dynamic time warping and gaussian distribution model for detecting emerging zero-day microarchitectural side-channel attacks. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 604–611. IEEE, 2020.

[33] Han Wang, Soheil Salehi, Hossein Sayadi, Avesta Sasan, Tinoosh Mohsenin, PD Sai Manoj, Setareh Rafatirad, and Houman 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)*, pages 1–4. IEEE, 2021.

[34] Han Wang, Hossein Sayadi, Sai Manoj Pudukotai Dinakarrao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Enabling micro ai for securing edge devices at hardware level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 11(4):803–815, 2021.

[35] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, Tinoosh Mohsenin, and Houman Homayoun. Comprehensive evaluation of machine learning countermeasures for detecting microarchitectural side-channel attacks. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, pages 181–186, 2020.

[36] Han Wang, Hossein Sayadi, Tinoosh Mohsenin, Liang Zhao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Hybrid-shield: Accurate and efficient cross-layer countermeasure for run-time detection and mitigation of cache-based side-channel attacks. IEEE/ACM International Conference on Computer-Aided Design, 2020.

[37] Han Wang. Leaked-web: Accurate and efficient machine learning-based website fingerprinting attack through hardware performance counters. *arXiv preprint arXiv:2110.01202*, 2021.

[38] Han Wang, Syed Mahbub Hafiz, Kartik Patwari, Chen-Nee Chuah, Zubair Shafiq, and Houman Homayoun. Stealthy inference attack on dnn via cache-based side-channel attacks. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1515–1520. IEEE, 2022.

[39] Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Sourangshu Bhattacharya. Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks. *IACR Cryptol. ePrint Arch.*, 2017:564, 2017.

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

[41] Jonas Depoix and Philipp Altmeyer. Detecting spectre attacks by identifying cache side-channel attacks using machine learning. *Advanced Microkernel Operating Systems*, 75, 2018.

[42] Maria Mushtaq, Jeremy Bricq, Muhammad Khurram Bhatti, Ayaz Akram, Vianney Lapotre, Guy Gogniat, and Pascal Benoit. Whisper: A tool for run-time detection of side-channel attacks. *IEEE Access*, 8:83871–83900, 2020.

[43] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. Stealthmem: System-level protection against cache-based side channel attacks in the cloud. In *USENIX Security symposium*, pages 189–204, 2012.

[44] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[45] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. *Hasp@ isca*, 10(1), 2013.

[46] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[47] Sergei Arnautov et al. Scone: Secure linux containers with intel sgx. In *OSDI*, volume 16, pages 689–703, 2016.

[48] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[49] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel {SGX} kingdom with transient out-of-order execution. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 991–1008, 2018.

[50] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, page 2. ACM, 2017.

[51] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2421–2434. ACM, 2017.

[52] Bart Coppens, Ingrid Verbauwhede, Koen De Bosschere, and Bjorn De Sutter. Practical mitigations for timing-based side-channel attacks on modern x86 processors. In *2009 30th IEEE Symposium on Security and Privacy*, pages 45–60. IEEE, 2009.

[53] Leonid Domnitser and et.al. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM TACO*, 2012.

[54] Zeyu Mi and et.al. Cpu elasticity to mitigate cross-vm runtime monitoring. *IEEE Transactions on Dependable and Secure Computing*, 2018.

[55] Tatu Ylonen, Chris Lonvick, et al. The secure shell (ssh) protocol architecture, 2006.

[56] Rishab Nithyanand, Xiang Cai, and Rob Johnson. Glove: A bespoke website fingerprinting defense. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 131–134, 2014.

[57] Inc. The Tor Project. The tor browser. *https://www.torproject.org/projects/torbrowser.html.en.*

[58] Suman Jana and Vitaly Shmatikov. Memento: Learning secrets from process footprints. In *2012 IEEE Symposium on Security and Privacy*, pages 143–157. IEEE, 2012.

[59] Raphael Spreitzer, Simone Griesmayr, Thomas Korak, and Stefan Mangard. Exploiting data-usage statistics for website fingerprinting attacks on android. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, pages 49–60, 2016.

[60] Anatoly Shusterman and et.al. Website fingerprinting through the cache occupancy channel and its real world practicality. *IEEE TDSC*, 2020.

[61] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. *arXiv preprint arXiv:2008.04480*, 2020.

[62] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. Fpguard: Detection and prevention of browser fingerprinting. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 293–308. Springer, 2015.

[63] Berk Gulmezoglu, Andreas Zankl, Thomas Eisenbarth, and Berk Sunar. Perfweb: How to violate web privacy with hardware performance events. In *European Symposium on Research in Computer Security*, 2017.

[64] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*, pages 820–830, 2015.

[65] Mengjia Yan and et al. Cache telepathy: Leveraging shared resource attacks to learn dnn architectures. In *29th Security Symposium USENIX Security 20)*, pages 2003–2020, 2020.

[66] Weizhe Hua and et al. Reverse engineering convolutional neural networks through side-channel information leaks. In *2018 55th DAC*, pages 1–6. IEEE, 2018.

[67] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[68] Gaofeng Dong, Ping Wang, Ping Chen, Ruizhe Gu, and Honggang Hu. Floating-point multiplication timing attack on deep neural network. In *2019 IEEE International Conference on Smart Internet of Things (SmartIoT)*, pages 155–161. IEEE, 2019.

[69] Han Wang, Hossein Sayadi, Tinoosh Mohsenin, Liang Zhao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis. In *DATE'20*. IEEE, 2020.

[70] Ferdinand Brasser and et al. Advances and throwbacks in hardware-assisted security: Special session. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, page 15. IEEE Press, 2018.

[71] Sanjeev Das, Jan Werner, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. Sok: The challenges, pitfalls, and perils of using hardware performance counters for security.

[72] Boyou Zhou, Anmol Gupta, Rasoul Jahanshahi, Manuel Egele, and Ajay Joshi. Hardware performance counters can detect malware: Myth or fact? In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 457–468. ACM, 2018.

[73] Hossein Sayadi, Nisarg Patel, Sai Manoj PD, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[74] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. *Retrieved from School of Computer Science Adelaide: http://cs. adelaide. edu. au/˜ yval/Mastik*, 2016.

[75] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. Xlate: https://www.vusec.net/projects/xlate/.

[76] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[77] Lexiang Ye and Eamonn Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956. ACM, 2009.

[78] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[79] Jessica Lin and Yuan Li. Finding structural similarity in time series data using bag-of-patterns representation. In *International Conference on Scientific and Statistical Database Management*, pages 461–477. Springer, 2009.

[80] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing sax: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[81] TIANWEI Zhang and RUBY B Lee. Secure cache modeling for measuring side-channel leakage. *Technical Report, Princeton University*, 2014.

[82] Matthew R Guthaus, Jeffrey S Ringenberg, Dan Ernst, Todd M Austin, Trevor Mudge, and Richard B Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Fourth WWC-4*. IEEE, 2001.

[83] Ben D Fulcher and Nick S Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014.

[84] Rich Caruana and Dayne Freitag. Greedy attribute selection. In *Machine Learning Proceedings 1994*, pages 28–36. Elsevier, 1994.

[85] Han Wang, Hossein Sayadi, Tinoosh Mohsenin, Liang Zhao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Mitigating cache-based side-channel attacks through randomization: A comprehensive system and architecture level analysis. In *DATE*. IEEE, 2020.

[86] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18. ACM, 2017.

[87] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, Tinoosh Mohsenin, and Houman Homayoun. Comprehensive evaluation of machine learning countermeasures for detecting microarchitectural side-channel attacks. In *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, page 181–186.

[88] Hossein Sayadi, Han Wang, Tahereh Miari, Hosein Mohammadi Makrani, Mehrdad Aliasgari, Setareh Rafatirad, and Houman Homayoun. Recent advancements in microarchitectural security: Review of machine learning countermeasures. In *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 949–952. IEEE, 2020.

[89] Fairuz Amalina Narudin et al. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, pages 343–357, 2016.

[90] Sai Manoj Pudukotai Dinakarrao, Sairaj Amberkar, Sahil Bhat, Abhijitt Dhavlle, Hossein Sayadi, Avesta Sasan, Houman Homayoun, and Setareh Rafatirad. Adversarial attack on microarchitectural events based malware detectors. In *Proceedings of the 56th Annual Design Automation Conference 2019*, pages 1–6, 2019.

[91] Robert Lim, David Carrillo-Cisneros, W Alkowaileet, and I Scherson. Computationally efficient multiplexing of events on hardware counters. In *Linux Symposium*, pages 101–110. Citeseer, 2014.

[92] Dehao Chen, Neil Vachharajani, Robert Hundt, Shih-wei Liao, Vinodha Ramasamy, Paul Yuan, Wenguang Chen, and Weimin Zheng. Taming hardware event samples for fdo compilation. In *Proceedings of the 8th annual IEEE/ACM international symposium on Code generation and optimization*, pages 42–52. ACM, 2010.

[93] Hossein Sayadi and et al. Stealthminer: Specialized time series machine learning for run-time stealthy malware detection based on microarchitectural features. In *GLSVLSI'20*, page 175–180, 2020.

[94] Junaid Nomani and Jakub Szefer. Predicting program phases and defending against side-channel attacks using hardware performance counters. In *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy*, page 9. ACM, 2015.

[95] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[96] Eamonn J Keogh and Michael J Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM international conference on data mining*, pages 1–11. SIAM, 2001.

[97] Tesla autopilot. *https://www.tesla.com/ autopilot.*

[98] Inside a self-driving uber. *https://www.infoq.com/presentations/uber-self-driving-software.*

[99] Waymo's autonomous fleet has intel inside. *https://www.electronicdesign.com/automotive/waymo-s-autonomous-fleet-has-intel-inside.*

[100] Baidu apollo. *https://www.electronicdesign.com/markets /automotive/article/21119589/xilinx-soc-fpga-powers-baidus-apollo-driverless-platform.*

[101] Andrew Bacha, Cheryl Bauman, Ruel Faruque, Michael Fleming, Chris Terwelp, Charles Reinholtz, Dennis Hong, Al Wicks, Thomas Alberi, David Anderson, et al. Odin: Team victortango's entry in the darpa urban challenge. *Journal of field Robotics*, 25(8):467–492, 2008.

[102] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA urban challenge: autonomous vehicles in city traffic*, volume 56. springer, 2009.

[103] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[104] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *The international Journal of robotics research*, 22(12):985–1003, 2003.

[105] Path planning. In *https://github.com/ser94mor/path-planning.*

[106] Perf. In *https://perf.wiki.kernel.org/index.php/Main Page.*

[107] Adrian Tang, Simha Sethumadhavan, and Salvatore J Stolfo. Unsupervised anomaly-based malware detection using hardware features. In *RAID'14*, pages 109–129. Springer, 2014.

[108] Meltem Ozsoy, Caleb Donovick, Iakov Gorelik, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Malware-aware processors: A framework for efficient online malware detection. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 651–661. IEEE, 2015.

[109] Khaled N Khasawneh, Meltem Ozsoy, Caleb Donovick, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Ensemble learning for low-level hardware-supported malware detection. In *International Symposium on Recent Advances in Intrusion Detection*, pages 3–25. Springer, 2015.

[110] Baljit Singh, Dmitry Evtyushkin, Jesse Elwell, Ryan Riley, and Iliano Cervesato. On the detection of kernel-level rootkits using hardware performance counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 483–493, 2017.

[111] Smartphone Market Share. Available online:. https://www.idc.com/promo/smartphone-market-share/os.

[112] Mcafee labs threats report, mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-apr-2021.pdf. April 2021.

[113] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-tenant side-channel attacks in paas clouds. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 990–1003. ACM, 2014.

[114] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. S $ a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes. In *2015 IEEE Symposium on Security and Privacy*, pages 591–604. IEEE, 2015.

[115] Grégoire Jacob, Hervé Debar, and Eric Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in computer Virology*, 4(3):251–266, 2008.

[116] Markus Miettinen, Samuel Marchal, Ibbad Hafeez, N Asokan, Ahmad-Reza Sadeghi, and Sasu Tarkoma. Iot sentinel: Automated device-type identification for security enforcement in iot. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2177–2184. IEEE, 2017.

[117] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. On the feasibility of online malware detection with performance counters. *ACM SIGARCH Computer Architecture News*, 41(3):559–570, 2013.

[118] A. Mosenia and N. K. Jha. A comprehensive study of security of internet-of-things. *IEEE Transactions on Emerging Topics in Computing*, 5(4):586–602, Oct 2017.

[119] Anand Mudgerikar, Puneet Sharma, and Elisa Bertino. Edge-based intrusion detection for iot devices. *ACM Transactions on Management Information Systems (TMIS)*, 11(4):1–21, 2020.

[120] Intel Intel. ia-32 architectures software developer's manual, volume 3b: System programming guide. *Part*, 1:64, 2007.

[121] Nisarg Patel, Avesta Sasan, and Houman Homayoun. Analyzing hardware based malware detectors. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2017.

[122] Zhihan Lv. Security of internet of things edge devices. *Software: Practice and Experience*, 2020.

[123] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[124] Venkatesh Gauri Shankar, Gaurav Somani, Manoj Singh Gaur, Vijay Laxmi, and Mauro Conti. Androtaint: An efficient android malware detection framework using dynamic taint analysis. In *2017 ISEA Asia security and privacy (ISEASP)*, pages 1–13. IEEE, 2017.

[125] Niall McLaughlin, Jesus Martinez del Rincon, BooJoong Kang, Suleiman Yerima, Paul Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, et al. Deep android malware detection. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 301–308, 2017.

[126] Min Yeol Lim, Allan Porterfield, and Robert Fowler. Softpower: fine-grain power estimations using performance counters. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 308–311, 2010.

[127] Han Wang, Setareh Rafatirad, and Houman Homayoun. A+ tuning: Architecture+ application auto-tuning for in-memory data-processing frameworks. In *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 163–166. IEEE, 2019.

[128] Robert O'Callahan, Chris Jones, Nathan Froyd, Kyle Huey, Albert Noll, and Nimrod Partush. Engineering record and replay for deployability. In *2017 {USENIX} Annual Technical Conference*, pages 377–389, 2017.

[129] GL Tsafack Chetsa, Laurent Lefèvre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. Exploiting performance counters to predict and improve energy performance of hpc systems. *Future Generation Computer Systems*, 36:287–298, 2014.

[130] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman 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*, pages 1–9, 2020.

[131] Arm cortex-a7. In *https://developer.arm.com/documentation/ddi0464 /d/Performance-Monitoring-Unit/About-the-Performance-Monitoring-Unit*.

[132] *https://perf.wiki.kernel.org/index.php*. Last accessed: 20-Feb-2019.

[133] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference*, volume 710, 1999.

[134] James Reinders. Vtune performance analyzer essentials. *Intel Press*, 2005.

[135] Android Open Source Project: Simpleperf.

[136] Virustotal intelligence service, 2018. Last accessed: 20-Feb-2019.

[137] Hossein Sayadi, Hosein Mohammadi Makrani, Sai Manoj Pudukotai Dinakarrao, Tinoosh Mohsenin, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2smart: A two-stage machine learning-based approach for run-time specialized hardware-assisted malware detection. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 728–733. IEEE, 2019.

[138] https://venturebeat.com/2018/10/01/xilinx-will-use-arm-cores-in-fpga chips/. Xilinx will use arm cores in fpga chips. In *VentureBeat*.

[139] Ilya K Ganusov, Mahesh A Iyer, Ning Cheng, and Alon Meisler. Agilex™ generation of intel® fpgas. In *2020 IEEE Hot Chips 32 Symposium (HCS)*, pages 1–26. IEEE Computer Society, 2020.

[140] Part Guide. Intel® 64 and ia-32 architectures software developer's manual. *Volume 3B: System programming Guide, Part*, 2, 2011.

[141] Tien-Fu Chen and Jean-Loup Baer. Effective hardware-based data prefetching for high-performance processors. *IEEE transactions on computers*, 44(5):609–623, 1995.

[142] Antonio González and et.al. A data cache with multiple caching strategies tuned to different types of locality. In *International Conference on Supercomputing*, 1995.

[143] Steven P Vanderwiel and David J Lilja. Data prefetch mechanisms. *ACM Computing Surveys (CSUR)*, 32(2):174–199, 2000.

[144] Kyle J Nesbit and James E Smith. Data cache prefetching using a global history buffer. In *10th HPCA'04*.

[145] Kyle J Nesbit and et.al. Ac/dc: An adaptive data cache prefetcher. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*.

[146] John WC Fu, Janak H Patel, and Bob L Janssens. Stride directed prefetching in scalar processors. In *Proceedings the 25th Annual International Symposium on Microarchitecture MICRO 25*, pages 102–110. IEEE, 1992.

[147] Disclosure of h/w prefetcher control on some intel processors. In *https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors*.

[148] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–17. Springer, 2009.

[149] Chester Rebeiro, David Selvakumar, and ASL Devi. Bitslice implementation of aes. In *International Conference on Cryptology and Network Security*, pages 203–212. Springer, 2006.

[150] Robert Könighofer. A fast and cache-timing resistant implementation of the aes. In *Cryptographers' Track at the RSA Conference*, pages 187–202. Springer, 2008.

[151] Jonghyeon Cho, Taehun Kim, Soojin Kim, Miok Im, Taehyun Kim, and Youngjoo Shin. Real-time detection for cache side channel attack using performance counter monitor. *Applied Sciences*, 10(3):984, 2020.

[152] M Asim Mukhtar, Maria Mushtaq, M Khurram Bhatti, Vianney Lapotre, and Guy Gogniat. Flush+ prefetch: A countermeasure against access-driven cache-based side-channel attacks. *Journal of Systems Architecture*, 104:101698, 2020.

[153] John L Henning. Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17, 2006.

[154] Arvind Narayanan, Hristo Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *2012 IEEE Symposium on Security and Privacy*, pages 300–314. IEEE, 2012.

[155] Vera Rimmer, Davy Preuveneers, Marc Juarez, Tom Van Goethem, and Wouter Joosen. Automated website fingerprinting through deep learning. *arXiv preprint arXiv:1708.06376*, 2017.

[156] Xiapu Luo, Peng Zhou, Edmond WW Chan, Wenke Lee, Rocky KC Chang, and Roberto Perdisci. Httpos: Sealing information leaks with browser-side obfuscation of encrypted flows. In *NDSS*, volume 11, 2011.

[157] Jo Booth. *Not so incognito: Exploiting resource-based side channels in JavaScript engines*. PhD thesis, 2015.

[158] Pavel Lifshits, Roni Forte, Yedid Hoshen, Matt Halpern, Manuel Philipose, Mohit Tiwari, and Mark Silberstein. Power to peep-all: Inference attacks by malicious batteries on mobile devices. *Proceedings on Privacy Enhancing Technologies*, 2018(4):141–158, 2018.

[159] Hyungsub Kim, Sangho Lee, and Jong Kim. Inferring browser activity and status through remote monitoring of storage usage. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, 2016.

[160] Shane S Clark, Hossen Mustafa, Benjamin Ransford, Jacob Sorber, Kevin Fu, and Wenyuan Xu. Current events: Identifying webpages by tapping the electrical outlet. In *European Symposium on Research in Computer Security*, pages 700–717. Springer, 2013.

[161] Pepe Vila and Boris Köpf. Loophole: Timing attacks on shared event loops in chrome. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 849–864, 2017.

[162] Sangho Lee, Youngsok Kim, Jangwoo Kim, and Jong Kim. Stealing webpages rendered on your browser by exploiting gpu vulnerabilities. In *2014 IEEE Symposium on Security and Privacy*, pages 19–33. IEEE, 2014.

[163] Qing Yang, Paolo Gasti, Gang Zhou, Aydin Farajidavar, and Kiran S Balagani. On inferring browsing activity on smartphones via usb power analysis side-channel. *IEEE Transactions on Information Forensics and Security*, 12(5):1056–1066, 2016.

[164] Hosein Mohammadi Makrani, Hossein Sayadi, Devang Motwani, Han Wang, Setareh Rafatirad, and Houman Homayoun. Energy-aware and machine learning-based resource provisioning of in-memory analytics on cloud. In *Proceedings of the ACM Symposium on Cloud Computing*, pages 517–517, 2018.

[165] Han Wang. Survey on performance analysis of virtualized systems. 2019.

[166] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Dreal: Detecting side-channel attacks at real-time using low-level hardware features. Technical report, University of California Davis United States, 2020.

[167] Han Wang, Hossein Sayadi, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. Hybridg: Hybrid dynamic time warping and gaussian distribution model for detecting emerging zero-day microarchitectural side-channel attacks. In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 604–611. IEEE, 2020.

[168] Han Wang, Hossein Sayadi, Gaurav Kolhe, Avesta Sasan, Setareh Rafatirad, and Houman 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)*, pages 648–655. IEEE, 2020.

[169] Han Wang, Hossein Sayadi, Avesta Sasan, PD Sai Manoj, Setareh Rafatirad, and Houman Homayoun. Machine learning-assisted website fingerprinting attacks with side-channel information: A comprehensive analysis and characterization. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 79–84. IEEE, 2021.

[170] Mohammadkazem Taram, Dean Tullsen, Ashish Venkat, Hossein Sayadi, Han Wang, Sai Manoj, and Houman Homayoun. Fast and efficient deployment of security defenses via context sensitive decoding. Technical report, University of California San Diego United States, 2019.

[171] Hoda Naghibijouybari, Ajaya Neupane, Zhiyun Qian, and Nael Abu-Ghazaleh. Rendered insecure: Gpu side channel attacks are practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2139–2153, 2018.

[172] Perf. In *https://perf.wiki.kernel.org/index.php/Main Page*.

[173] Muhammad Hilmi Kamarudin, Carsten Maple, Tim Watson, and Nader Sohrabi Safa. A logitboost-based algorithm for detecting known and unknown web attacks. *IEEE Access*, 5:26190–26200, 2017.

[174] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[175] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[176] Alexa Analitic. The top 500 sites on the web https://www.alexa.com/topsites. *Retrieved July*, 3:2017, 2020.

[177] Mark Hall et al. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[178] Statista report. In *https://www.statista.com/statistics/607716/worldwide-artificial-intelligence-market-revenues/*.

[179] The wall street journal. In *https://www.wsj.com/articles/the-economic-value-of-artificial-intelligence-1540568499*.

[180] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[181] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[182] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[183] Antônio H Ribeiro, Manoel Horta Ribeiro, Gabriela MM Paixão, Derick M Oliveira, Paulo R Gomes, Jéssica A Canazart, Milton PS Ferreira, Carl R Andersson, Peter W Macfarlane, Wagner Meira Jr, et al. Automatic diagnosis of the 12-lead ecg using a deep neural network. *Nature communications*, 11(1):1–9, 2020.

[184] Shayan Hassantabar, Mohsen Ahmadi, and Abbas Sharifi. Diagnosis and detection of infected tissue of covid-19 patients based on lung x-ray image using convolutional neural network approaches. *Chaos, Solitons & Fractals*, 140:110170, 2020.

[185] Liang Zhou and et al. Cyber-attack classification in smart grid via deep neural network. In *Proceedings of the 2nd International Conference on Computer Science and Application Engineering*, pages 1–5, 2018.

[186] Cheng-Bin Jin and et al. Real-time human action recognition using cnn over temporal images for static video surveillance cameras. In *Pacific Rim Conference on Multimedia*. Springer, 2015.

[187] Yuntao Liu and Ankur Srivastava. Ganred: Gan-based reverse engineering of dnns via cache side-channel. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 41–52, 2020.

[188] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[189] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[190] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[191] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[192] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[193] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th MALWAR)*, pages 11–20. IEEE, 2015.

[194] Flower dataset. In *https://storage.googleapis.com/download.tensorflow.org/ example_images/flower_photos.tgz*.