SPY-PMU: Side-Channel Profiling of Your Performance Monitoring Unit to Leak Remote User Activity

Md Kawser Bepary, Arunabho Basu, Sajeed Mohammad, Rakibul Hassan, Farimah Farahmandi, Mark Tehranipoor
Dept. of Electrical and Computer Engineering, University of Florida, Gainesville, Florida 32611, USA
Email: {mdkawser.bepary, arunabhobasu, mlnu, rhassan1}@ufl.edu, {farimah, tehranipoor}@ece.ufl.edu

Abstract*—The Performance Monitoring Unit (PMU), a standard feature in all modern computing systems, presents significant security risks by leaking sensitive user activities through microarchitectural event data. This work demonstrates the feasibility of remote side-channel attacks leveraging PMU data, revealing vulnerabilities that compromise user privacy and enable covert surveillance without physical access to the target machine. By analyzing the PMU feature space, we create distinct microarchitectural fingerprints for benchmark applications, which are then utilized in machine learning (ML) models to detect the corresponding benchmarks. This approach allows us to build a pre-trained model for benchmark detection using the unique microarchitectural fingerprints derived from PMU data. Subsequently, when an attacker remotely accesses the victim's PMU data, the pre-trained model enables the identification of applications used by the victim with high accuracy. In our proof-of-concept demonstration, the pre-trained model successfully identifies applications used by a victim when the attacker remotely accesses PMU data, showcasing the potential for malicious exploitation of PMU data. We analyze stress-ng benchmarks and build our classifiers using logistic regression, decision tree, k-nearest neighbors, and random forest ML models. Our proposed models achieve an average prediction accuracy of 98%, underscoring the potential risks associated with remote side-channel analysis using PMU data and emphasizing the need for more robust safeguards. This work underscores the urgent need for robust countermeasures to protect against such vulnerabilities and provides a foundation for future research in micro-architectural security.

***Index Terms*—Cybersecurity, Microarchitectural Security, Side Channel Analysis, Performance Monitoring Unit (PMU), Application Fingerprinting, User Activity Leakage**

I. Introduction

In the past decade, side-channel attacks have emerged as critical threats since they can leak sensitive data about security assets or user activity [1], [2]. Unlike traditional attacks, which exploit vulnerabilities in software or hardware, side-channel attacks exploit information leaked in the form of timing data, power consumption, electromagnetic emissions, etc which is inadvertently leaked by a computing system during normal operation. Studying the hardware behavior based on the subtle variations from the data leaked physically to its surroundings, attackers can infer security assets or user behavior, compromising the system integrity and privacy. These attacks are particularly insidious because they bypass traditional security measures designed to safeguard software or network layers, revealing a significant blind spot in modern defenses [2].

Performance Monitoring Units (PMUs), designed to provide low-level insights into microarchitectural events, have emerged as a powerful source of side-channel data. PMUs are capable of recording detailed patterns of CPU utilization, memory access, cache behavior, and other microarchitectural activities [3], [4]. While their intended use is to aid diagnostics and debugging, their granularity makes them attractive for malicious purposes. Studies have shown that PMU data can be exploited to infer higher-level applications through machine learning (ML) models trained on low-level microarchitectural behavior [5], [6]. This highlights the dual-edged nature of PMU functionality: beneficial for legitimate use but vulnerable to misuse by adversaries.

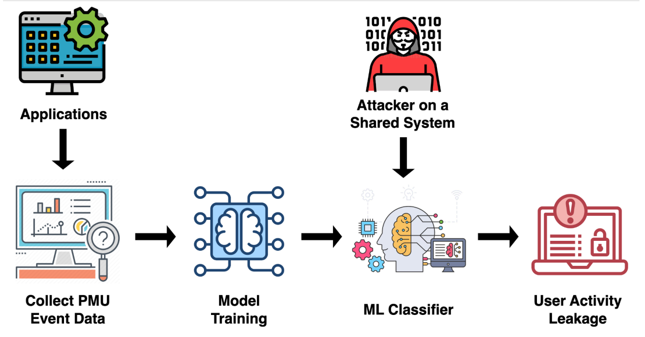


Fig. 1: Overview of the potential risks associated with PMU data in a side-channel attack

A particularly concerning dimension of PMU-based attacks is their potential for remote execution [7]. Attackers can exploit software vulnerabilities to gain unauthorized access to PMU data, enabling remote analysis and inference of user activity. This risk is amplified in cloud computing and multi-tenant environments where multiple users share the same physical hardware [8], [9]. Unlike traditional side-channel attacks that often require physical proximity to the target, PMU-based remote attacks can be executed covertly, leaving no forensic trace on the victim's system. Figure 1 provides an overview of the potential risks associated with PMU data in a side-channel attack, illustrating the threats posed by unauthorized access and misuse of PMU data. Recent work, such as speculative execution attacks [7], has demonstrated how PMU data can serve as a foundation for extracting sensitive user information, emphasizing the need for robust defenses against this class of attacks.

Although prior research has explored side-channel vulnerabilities using PMU data [4], [8], much of the focus has been on coarse-grained profiling or constrained environments. The potential for leveraging PMU data to execute remote attacks in real-world, multi-tenant settings remains underexplored. Additionally, prior works have primarily demonstrated benchmark classification, but have not addressed more impactful applications, such as inferring cryptographic keys or user credentials [5], [7]. Our work bridges this gap by presenting a proof of concept that demonstrates the feasibility of remote user activity detection using PMU profiling in practical environments.

To this end, we systematically collected PMU data across several CPU benchmarks, representing a wide range of microarchitectural events. Through detailed analysis, we observed distinctive patterns that correlate higher-level user activities with low-level hardware behavior. By leveraging these insights, we trained ML classifiers to detect user activities with high accuracy, achieving up to 99% prediction accuracy. Our contributions are as follows:

- **Profiling Microarchitectural Behavior**: We developed application-specific microarchitectural profiles based on PMU data, offering detailed insights into user activity.
- **Analysis of PMU Events**: We identified key PMU events that significantly contribute to user activity classification, providing a foundation for efficient feature selection.
- **Remote Activity Classification**: We demonstrated the feasibility of using PMU data to classify user activities remotely in shared computing environments.
- **Call for Security Measures**: We emphasized the urgent need for countermeasures to mitigate PMU-based side-channel attacks, such as access restrictions and obfuscation techniques.

Our findings underscore the necessity of re-evaluating the security risks posed by PMU functionality in modern processors. While PMU data provides valuable diagnostic capabilities, its misuse presents a critical security challenge. By highlighting the feasibility of remote PMU-based attacks, our work not only raises awareness but also calls for a concerted effort to develop robust defenses that can prevent unauthorized exploitation of PMU data.

II. Background

A. Performance Monitoring Unit (PMU)

Performance Monitoring Units (PMUs) [9] are specialized hardware components embedded in modern processors that provide observability of micro-architectural events involved during program execution [9]. They play a great role in debugging, performance profiling and system optimization. PMUs are implemented using Model Specific Registers (MSR), which are programmable registers embedded in the CPU. They can be programmed to perform a specific task; in this case, the PMU is implemented by programming them to form simple counters that increment by 1 whenever a particular micro-architectural event is observed [10]. The processors ship with pre-programmed PMU events called native events, but it is possible to program our own custom events. The native events are robust enough for achieving the goal of our proof of concept.

Modern processors have PMUs that support recording of a wide range of micro-architectural events indicative of system performance, like cache misses, branch predictions, memory accesses and CPU cycles. These allow developers to monitor and analyze the interaction of their software with underlying hardware. For example, high cache miss rates would imply that the software is not utilizing memory hierarchies efficiently, prompting the developer to optimize their code. PMUs provide fine grained visibility into micro-architectural events. However, the same granular recording of low-level tasks, which is beneficial for developers, can be exploited by attackers to perform side-channel attacks (SCAs) [6], [9]. By analyzing the variations in cache behavior [11], memory accesses [12] or CPU branch patterns, an attacker can infer user activities [13]. This capability is particularly concerning in shared environments, such as multi-tenant cloud systems, where attackers can eavesdrop on a victim's workload by analyzing PMU data [14].

In recent years, PMUs have been increasingly exploited in SCAs where attackers analyzed the micro-architectural behavior of the target system to infer sensitive information, such as cryptographic keys [12] or user activity, often with surprising accuracy [9], [15]–[17]. The ability to carry out these attacks remotely, coupled with the difficulty of detecting them, amplifies their threat [8]. PMU-based attacks remain largely underexplored in terms of mitigation, underscoring the need for robust security mechanisms.

B. Remote Side-Channel Attacks

While traditional Side Channel Attacks rely on physical access to hardware or its proximity to be able to capture and measure physical attributes like power consumption, electromagnetic emission or noise generation during computing operation, remote SCA [18] involve a connected network and software interfaces. The threat model for remote SCA assumes that an attacker has privileged access to the target computer. An attacker can attain elevated privileges exploiting a multitude of vulnerabilities. Once privileged access is obtained, an attacker can obtain unauthorized remote to hardware resources like the PMUs. They can be remotely activated using system calls or APIs like Performance-API (PAPI) present in modern Operating Systems. Subsequently they can record micro-architectural events and exfiltrate the data to remote computer, achieving remote monitoring [19], [20]. These attacks are stealthy as they do not noticeably modify any system software or hardware and PMU data collection incurs very little performance overhead. The attacker's process can run in the background collecting PMU data and exfiltrate it for remote analysis leaving no traces on the target machine. These characteristics of a remote SCA using PMU data make it very hard to detect and mitigate.

This problem of remote SCA is of particular concern [21] in multi-tenant environments like cloud computing [22], [23], where multiple users share the same underlying physical hardware [24] and rely on virtualization to maintain isolation between processes. However PMU data is often accessible across Virtual Machines (VMs), which would allow an attacker to monitor the micro-architectural behavior involved in other tenant's workloads [14]. This vulnerability [25] can be exploited to infer application exectuion patterns, memory access behavior or even cryptographic operations [26], [27] occuring in another tenant's VM. Thus, with the increased usage of multi-tenant environments, it is critical to address the growing threat posed by remote SCA. Detection and mitigation of such attacks require robust security mechanisms that can limit unauthorized access to PMU data, obfuscate micro-architectural event process or its logging, insert appropriate amount of noise into PMU data so that SCA are less effective but the data is still usable for debugging and system optimization. Hardware based counter-measures like restricting PMU access during execution of sensitive operations or adopting secure processor designs that minimize side channel leakage could be pivotal for safeguarding against such attacks.

C. Related Work

Physical side-channel [1], [2] attacks have become a significant threat to computing systems. This is because without exploiting any complex vulnerability of the system, attackers can extract sensitive information from various physical phenomena [1]–[4], [13], [19] like electromagnetic radiation, power consumption, and cache behavior – that are inadvertently leaked by hardware components during normal operation. A wide range of techniques [7]–[9] has been developed to execute such attacks or detect and mitigate them. One of the seminal works in this domain [19] showed that built-in sensors on modern PCs can be used to perform physical side-channel attacks remotely with no physical access to the victim's PC. They proved that the electromagnetic radiation leaked into the surroundings is deterministic enough to accurately profile the victim's user activity and this can be achieved by just exploiting the inbuilt microphone sensor on the victim's computer which can pick up on the electromagnetic leakage.

In parallel, major developments have been made in side-channel analysis involving PMU counters. Prior works [8] have shown that PMU counters can be super effective in side channel analysis and attacks. They demonstrated that all existing cache-side channel attacks are possible using PMUs. As a proof of concept, they implement successful attacks using PMU data, which exploit transient execution vulnerabilities like ZombieLand. Using this attack, they were successfully able to beat the Intel SGX and extract the AES encryption key from within the SGX enclave, thus proving the efficacy of PMUs in SCA. They also demonstrated [9] a novel side-channel attack that uses PMU counters to record events during transient executions without rollback. Countermeasures against this type of attack are – 1) hardware changes to rollback PMU counters during transient executions; and 2) disabling PMU when Intel SGX is in use.

In a related area, thousands of non-documented PMU events [10] were found to exist that can be used to perform side-channel analysis or attacks. The register used to select PMU events supports 2^{16} unique combinations while only about 300 are documented by chip manufacturers. The authors proved that using non-documented patterns leads to non-documented PMUs which can be used to profile microarchitectural behavior with varying degrees of success.

A novel attack, [28] combines power analysis with transient execution attacks to enhance the extraction of sensitive data. A multi-dimensional attack like this power side channel attack is hard to detect and can leak sensitive data. Another novel attack [29] demonstrated that a very lightweight framework of just CSS and HTML can be used to execute cache-based side channel attacks in browsers. The most significant part of this attack is that its architecture is agnostic and it has the capability of bypassing default security configurations. [30] showed how power consumption data is directly related to CPU activity and can be used to extract cryptographic keys and other security assets through techniques like SPA and DPA. This power consumption data could be accessed without admin privileges at the time of the publication of this paper.

In addition to these novel attacks using side-channel analysis, there have also been major advances in detecting and mitigating side-channel leakage. For instance, [7] developed a framework to quantify side-channel leakage from different cache architectures. This is done by modeling cache behavior and identifying potential leaks. The quantified leakage across various cache architectures would be useful to build secure cache architectures that would be resistant to side-channel attacks on cache using PMUs. [31] designed a detection system that combines Intel Cache Monitoring Technology (CMT) and HPCs to identify cache-based side-channel attacks. The fine granularity of CMT data, combined with statistical methods like Gaussian Anomaly Detection provides high accuracy with minimal performance overhead. [32] demonstrated that thread-level monitoring HPCs can give more granular data which can be used to identify cache side-channel attacks with greater precision. It complements other HPC-based detection systems and increases their efficiency.

These studies highlight the ongoing arms race between attackers and defenders exploiting side channel analysis at the physical layer. The diverse approaches and ongoing challenges call for continuous advancement in identifying new vulnerabilities and making roadways into detection and mitigation mechanisms.

III. Threat Model

This study examines a Performance Monitoring Unit (PMU)-based side-channel attack in multi-tenant environments, such as cloud computing platforms, data centers, and virtualized systems, where multiple users share the same physical hardware. These setups inherently rely on virtualization and logical isolation between tenants, which may inadvertently expose shared resources to side-channel exploitation [22], [24]. We consider an attacker who leverages privileged access to monitor PMU data and deduce sensitive user activities or

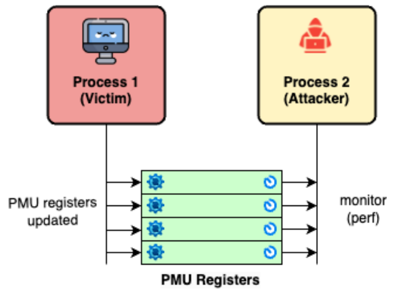


Fig. 2: Threat model of a PMU-based side-channel attack in a shared system.

extract security-critical information from co-resident victims on the shared system. Figure 2 illustrates the attack scenario.

A. Attacker Capabilities

The attacker in this threat model is assumed to possess the following capabilities:

- **Remote Access to PMU Data:** The attacker has remote access to the PMU data of the shared system. This access is facilitated through root or administrative privileges, allowing the attacker to monitor various microarchitectural events such as CPU utilization, memory accesses, and cache behavior. Such elevated privileges are realistic, as demonstrated in prior works that exploit vulnerabilities for privilege escalation [8], [9], [33].
- **No Physical Proximity:** The attack is executed remotely without the need for physical access to the victim's machine. The attacker executes a non-intrusive, low-overhead process or code that interfaces with the PMU, capturing microarchitectural event data while the victim performs sensitive tasks, such as entering credentials or executing cryptographic computations [34]. This capability underscores the stealthy nature of the attack, as PMU data collection leaves minimal traces on the victim's system.
- **Machine Learning-Based Inference:** The attacker pre-trains a machine learning model using PMU data from known workloads, including those involving sensitive user activities. This model is subsequently used to classify victim activities by correlating observed PMU traces with specific application patterns [17]. The attacker's expertise in feature selection, dimensionality reduction, and classifier training enables efficient inference despite the challenges posed by noise or multi-user environments.

Additionally, the attacker can scale their analysis to multiple users in a shared environment, leveraging the shared hardware to target several victims simultaneously [25]. This capability raises the stakes in multi-tenant setups, highlighting the insufficiency of current isolation mechanisms.

B. Attack Scenario

In this multi-tenant environment, the attacker and victim share the same physical hardware. The victim's tasks, such as online banking, browsing sensitive websites, or performing cryptographic operations, inadvertently generate distinguishable microarchitectural patterns captured by the PMU. The attacker runs a malicious process that collects PMU event data, either through APIs (e.g., Performance API or PAPI) or direct system calls, without the victim's awareness.

The attacker analyzes the PMU data to infer the victim's activities. For instance:

- By examining cache miss rates and memory access patterns, the attacker could determine the types of applications the victim is running.
- By observing subtle variations during cryptographic operations, the attacker could infer key-dependent execution patterns, potentially enabling cryptographic key extraction [7].

In multi-user scenarios, the attacker can utilize advanced machine learning techniques to distinguish individual users' activities despite noise and overlapping processes. This scalability further amplifies the attack's potential in real-world environments [25].

C. Security Implications

The implications of such an attack extend beyond mere activity detection and pose serious privacy and security risks:

- **Privacy Violation:** By classifying workloads, the attacker can infer sensitive user activities such as specific websites visited, keystrokes, or application usage. These insights could facilitate secondary attacks, such as phishing or financial fraud, compromising the victim's personal and financial data [34].
- **Cryptographic Key Leakage:** PMU-based monitoring of cryptographic operations increases the risk of key extraction through side-channel leakage. This is particularly dangerous in environments where encryption is relied upon for secure communications or sensitive data handling [7], [8].
- **Increased Vulnerability in Shared Environments:** Multi-tenant environments, common in cloud computing, exacerbate the risk by enabling attackers to observe and exploit shared hardware resources. The attacker's ability to scale the attack across multiple users increases the impact and difficulty of detection [22], [25].
- **Stealth and Detectability:** The benign nature of PMU data collection makes these attacks stealthy and challenging to detect. PMU data is routinely logged for diagnostics, and its collection incurs minimal performance overhead, allowing the attacker to operate without triggering suspicion [6].

This threat model emphasizes the urgency of addressing PMU-based side-channel vulnerabilities, particularly in scenarios involving shared hardware and multi-user environments. Robust countermeasures, such as restricting PMU access during sensitive operations, introducing noise into PMU data, and adopting secure hardware designs, are necessary to mitigate this emerging threat.

IV. Methodology

This section outlines the methodology for demonstrating the feasibility of a PMU-based remote side-channel attack. By systematically collecting PMU data, extracting relevant features, and employing machine learning classifiers, we show how user activity can be inferred in a shared computing environment. The methodology is organized into the following stages: Benchmark Execution, Data Collection, Feature Extraction, Classification Modeling, and Model Evaluation. Figure 3 provides an overview of the process.

A. Benchmark Applications Execution

The first stage involved collecting PMU data for a set of benchmark applications. Stress-ng benchmarking suite [35] was chosen for our work because it has a diverse set of benchmark applications that can cover a wide range of microarchitectural events. The benchmarks stress different parts of CPU and memory subsystems by performing a range of computational tasks indicative of common workloads. These chosen benchmarks can be broadly categorized as follows :

- **CPU and Memory-Intensive Benchmarks:** These include *cpu*, *cache*, *malloc*, *matrix*, and *bigheap*. These benchmarks stress the arithmetic logic units (ALUs), cache hierarchies, and memory allocation processes. For instance, *cpu* tests the processor's computational capacity by executing a series of arithmetic operations, while *cache* focuses on the efficiency of the cache subsystem, and *malloc* and *bigheap* stress memory allocation and usage patterns.
- **I/O-Related Benchmarks:** Benchmarks like *aio*, *io*, *sendfile*, and *hdd* are designed to test the system's performance under different I/O loads. These benchmarks simulate disk and network operations, such as asynchronous I/O (*aio*) and file sending (*sendfile*), providing insights into the processor's handling of I/O operations and related PMU events.
- **Process Management and Security Benchmarks:** These include *fork*, *exec*, *pthread*, *crypt*, and *sigsegv*, which focus on system call handling, process management, and security operations. For example, *fork* and *exec* simulate process creation and execution, critical for understanding the overheads and efficiency of the operating system's process management routines. *crypt* evaluates the system's ability to handle cryptographic operations, providing a security-focused perspective on PMU data.
- **System and Resource Management Benchmarks:** These benchmarks include *context*, *schedpolicy*, *schedprio*, and *membarrier*, which assess how the system manages resources, context switching, and scheduling. These tests provide insights into the performance overheads associated with multitasking and resource contention, revealing how these factors impact overall system performance.

This diverse selection of benchmarks ensures comprehensive coverage of the processor's behavior under different types of stress, allowing us to capture a wide array of PMU event signatures that are crucial for accurate classification.

B. Data Collection Process

The PMU data was recorded over multiple iterations of running the same benchmarking application for a regular interval of time, 10 seconds. The Intel architecture supports recording four PMU events simultaneously, and thus the PMU events list was further broken down into batches of four and each batch was measured multiple times for each benchmarking application so that we can reduce noise in the data preprocessing stage and get a more accurate representation of the data. Additionally, the cache was flushed between two instances of data recording to ensure consistency.

Algorithm 1 presents the PMU data collection process for *stress-ng* benchmarks using the *perf* utility. We developed a testing automation script involving two nested for loops to traverse the entire list of benchmarks and list of PMU events and record them. At first, the script goes over the list of benchmarks, and for each benchmarking application on the list, it creates a new directory and a new logfile for keeping records. Considering one benchmark at a time, it goes over the list of PMU events segregated in batches of four and selects one batch consecutively, flushes the cache and then it runs the benchmark and monitors the selected batch of PMU events simultaneously for an interval of 10 seconds. After its done, it pipes the output to a logfile and repeats the same steps for the next batch of PMU events. After its done with all the batches of the PMU events, it moves on to the next benchmarking application and does the same.

Algorithm 1 PMU Data Collection for Stress-ng Benchmarks

```
Require: stress-ng benchmarks, perf event_groups
Ensure: All benchmarks are executed and their PMU data is logged
 1: $B \leftarrow$ list of benchmarks
 2: $E \leftarrow$ list of event group arrays
 3: $D \leftarrow$ directory for logs
 4: Create directory $D$
 5: for $i = 1$ to length$(B)$ do
 6:     $d_i \leftarrow$ directory for $B[i]$
 7:     $L_i \leftarrow$ log file for combined data of $B[i]$
 8:     Create directory $d_i$
 9:     for $j = 1$ to length$(E[i])$ do
10:         Flush system caches
11:         $G_{ij} \leftarrow$ event group $E[i][j]$
12:         Run benchmark $B[i]$ stressing system with $G_{ij}$
13:         Monitor and log PMU events of $G_{ij}$ to $L_i$
14:     end for
15:     Log completion for $B[i]$
16: end for
```

C. Datasets

We systematically collected performance data across two structured datasets:

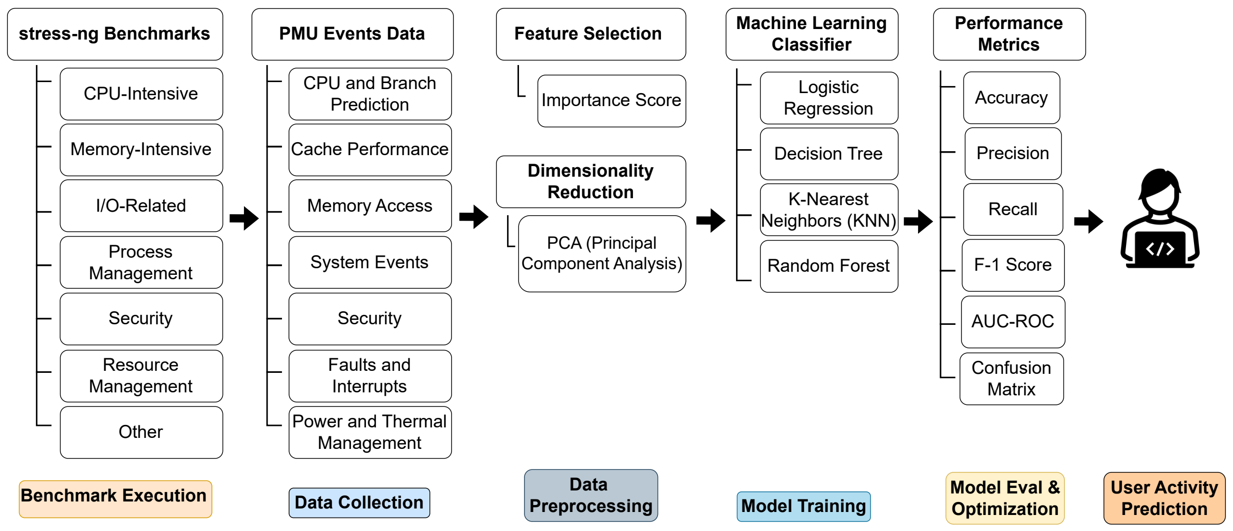


Fig. 3: Overview of the proposed methodology for PMU-based remote side-channel attack

- **Dataset 1:** Consists of 10 benchmarks and 20 PMU events. This smaller dataset was used to refine preprocessing, feature selection, and hyperparameter tuning.
- **Dataset 2:** Comprises 123 benchmarks and 81 PMU events, providing a comprehensive dataset to evaluate the scalability and robustness of our models.

Dataset 1 focused on core functionalities, such as cache and memory performance, while Dataset 2 extended coverage to broader system behaviors. This incremental approach allowed us to develop and test our framework effectively. For Dataset 1, the benchmarks such as *cache, vm, fork, io, clock, malloc*, and others were specifically chosen for their capacity to stress-test subsystems like the CPU and memory, providing valuable insights into performance characteristics vulnerable to side-channel attacks. 20 PMU events selected for Dataset 1 were the most relevant micro-architectural events involved in popular computational tasks. The selection of PMU events is further detailed in the following subsection.

D. Feature Extraction

After the PMU data was collected, the next stage involved feature extraction and feature engineering so that machine learning classifier models could be trained in the subsequent stage.

- **Selection of PMU Events**: PMUs on modern processors can monitor a wide range of microarchitectural events. These events form the feature space of our machine learning model. The intel processor we used in our test setup has native support for recording of over 150 micro-architectural events [36]. Majority of these events are not involved in day to day computational tasks, which necessitates the selection of PMU events that can record micro-architectural events involved in the most popular computational tasks. Accordingly, the most relevant PMU events [37] were selected and grouped as shown in Table I. These events also cover a wide range of microarchitectural behaviors allowing us to build a comprehensive microarchitectural behavioral profile.

TABLE I: Selected PMU Events

PMU Event Category	PMU Events
CPU and Branch Prediction	`branches`, `branch-misses`, `bus-cycles`, `branch-instructions`
Cache Performance	`cache-misses`, `cache-references`, `L1-dcache-loads`, `L1-dcache-load-misses`, `LLC-loads`, `LLC-load-misses`
Memory Access	`dTLB-loads`, `dTLB-load-misses`, `mem-loads`, `mem-stores`
System Events	`context-switches`, `cpu-clock`, `task-clock`, `cpu-migrations`
Faults and Interrupts	`alignment-faults`, `major-faults`, `minor-faults`, `page-faults`
Power and Thermal Management	`power/energy-cores`, `power/energy-pkg`, `msr/cpu_thermal_margin`, `cstate_pkg/c6-residency`
Uncore Events	`uncore_imc/data_reads`, `uncore_imc/data_writes`, `uncore_clock/clockticks`, `l1d.replacement`

- **Feature Engineering :** Next, the PMU data was preprocessed to ensure our machine learning model achieves high accuracy.
  - **Mean** : The data over several iterations was averaged to reduce noise and inconsistencies.
  - **Normalize** : The data was also normalized to a common scale so that the model focused on patterns of behavior rather than the absolute numbers.
  - **Uniform sampling :** Resampling was also done to maintain uniform time intervals.
  - **Principal Component Analysis:** With a high number of PMU events as input features, we ran the

risk of our classifier model overfitting the data and performing poorly on test sample because of a lack of generalization. So, it was necessary to reduce the dimension of features the classifier trains on. This was achieved using principal component analysis, which reduced the dimensions of the data while preserving the variance. The covariance matrix was calculated for the entire feature space consisting of PMU events, and it gave us the principal components which maintained the highest amount of covariance. The original feature space was subsequently reduced using the principal components only [1], [38].

E. Classification Modeling

After the PMU data was extracted and preprocessed, several machine learning classifiers were trained using this data. Multiple classifiers were used because each of them offers different strengths and trade-offs, thus allowing us to do a comparative study of which classifier works best for this type of data. The python package scikit-learn [39] was used to implement these classifiers. The classifiers used are as follows:

- **Logistic Regression :** We started with Logistic Regression, a simple yet effective linear classifier that was able to model the probability of a given application based on input features [40]. The model computes the probability p that a given set of features belongs to a particular category (positive class) using the logistic function:

$$p = \frac{1}{1+e^{-z}} \tag{1}$$

where z is defined as the linear combination of the input features:

$$z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n \tag{2}$$

Here $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients and $x_1, x_2, \ldots, x_n$ represent the normalized PMU event counts. It provided valuable insights into the linear separability of PMU event data and formed the baseline for the performance of more complex classifiers [40].

- **Decision Tree Classifier:** We implemented a Decision Tree Classifier to capture the non-linear relationships within the data. This model constructs a tree-like graph of decisions, where each node represents a feature split that best segregates the data into classes based on the impurity measures. The decision at each node n is based on a feature x_i and a threshold τ, forming the rule:

$$x_i \leq \tau \tag{3}$$

This approach effectively illustrates how different PMU events contribute to distinguishing between applications. The interpretability of the decision tree is especially valuable as it provides a clear visualization of feature importance and decision paths, which is crucial for understanding the underlying mechanisms of application detection [40].

- **K-Nearest Neighbors (KNN):** K-Nearest Neighbors algorithm was implemented next, which classifies samples based on the influence of their neighbors. This method considers both the distance to the nearest k neighbors and their respective influence on the classification decision. The decision rule for a given sample x is determined by the majority vote among its k nearest neighbors, formalized as:

$$y = \text{mode}\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\} \tag{4}$$

where y_{i_j} represents the class of the j-th nearest neighbor to x. This method is particularly robust in scenarios where the decision boundary is non-linear. By manipulating the parameter k, the number of neighbors considered, we tailored the model's sensitivity to accurately reflect the diverse distributions of PMU data [40].

- **Random Forest Classifier:** The Random Forest classifier combines several decision trees to form a more robust prediction model. Each tree in the ensemble is independently trained on a randomly selected subset of the data and features, using the following model:

$$\hat{y} = \frac{1}{N}\sum_{n=1}^{N} h_n(x) \tag{5}$$

where N is the number of trees, $h_n(x)$ is the prediction of the n-th tree, and x is the input feature vector. This methodology enhances the generalizability of the model by averaging the predictions, which effectively reduces the risk of overfitting typically associated with single decision trees. The use of random subsets for training each tree—known as bootstrap aggregating or bagging—ensures diversity in the model, contributing to its robust performance across various sets of PMU data [41].

Each classifier was trained on Dataset 1 and Dataset 2, with hyperparameters tuned for optimal performance.

F. Model Evaluation

Each of the models was evaluated using cross-validation with a training and test split of 80/20 where the classifier was trained on 80% of data set marked as training set and the trained model was tested against the 20% of dataset marked as test set. Models were evaluated using a combination of metrics to provide a comprehensive assessment:

- **Accuracy :** Accuracy measures the proportion of correct predictions out of the total predictions made. It is useful when the classes are balanced.
- **Precision :** Precision indicates how many of the predicted positive instances are actually positive. It is useful when minimizing false positives is important.
- **Recall :** Recall (or sensitivity) measures how many actual positive instances are correctly predicted. It is useful when minimizing false negatives is crucial.

- **F1-score :** The F1-score is the harmonic mean of precision and recall, providing a balance between the two, especially in cases of class imbalance.
- **AUC-ROC :** The Area Under the Curve of the Receiver Operating Characteristic (AUC-ROC) is a performance measurement for classification problems at various threshold settings. The ROC is a probability curve, and AUC represents the degree to which the model is capable of distinguishing between classes. An AUC of 1 represents a perfect model, while an AUC of 0.5 suggests no discriminative ability, equivalent to random guessing.
- **Confusion Matrix :** The confusion matrix is a table that describes the performance of a classification model by showing the true positives, true negatives, false positives, and false negatives. It provides insight into the types of errors made by the model.

These metrics help evaluate the performance of a classifier from different perspectives, especially in imbalanced datasets like ours where accuracy alone may be misleading.

V. Experimental Results

In this section, we present the results of our proof-of-concept study demonstrating the feasibility of PMU-based benchmark classification to infer user activity. The experiments include data preprocessing, feature selection, dimensionality reduction, and the evaluation of machine learning models on two datasets of varying complexity.

A. Experimental Setup

The PMU data collection was conducted on a system equipped with a modern multi-core Intel i7-1065G7 processor. A modern Intel processor was selected for its widespread use in both consumer and enterprise systems, thus making it a relevant platform for security analysis. The Operating system Ubuntu 22.04.4 LTS was used for its robust support, documentation, and granularity of control, which allowed us to fine-tune a couple of parameters to reduce noise in our data and get consistent results. The OS was configured to allow privileged access to PMU data, and it was recorded using the *perf* tool. Additionally, perf adds negligible computational overhead, thus reducing the noise in our measured data.

B. Data Preprocessing and Feature Analysis

*1) **Feature Selection and Importance**:* The initial step in our data analysis process involved selecting the most relevant features based on their importance scores derived from a Random Forest classifier. This model provides a robust mechanism for feature evaluation, as it computes importance based on how much each feature decreases the impurity of a split. Features are considered more important if they consistently improve the model's accuracy when included in trees. A threshold was applied to retain only the most impactful PMU events, reducing redundancy and improving efficiency. For Dataset 1, 17 features were selected from an initial set of 20 using a threshold of 0.02. For Dataset 2, the threshold of 0.01 narrowed the features from 81 to 45, as shown in Figure 4. These selected features highlighted key microarchitectural events, such as *L1-dcache-load-misses*, *cache-references*, and *L1-dcache-loads*, which played a critical role in distinguishing benchmark behaviors.

Figure 5 illustrates the dynamic behavior of these features through time-series and distribution plots. Time-series plots (top row) reveal temporal variations across benchmarks, while distribution plots (bottom row) showcase the variability and clustering of PMU events for different benchmarks.

TABLE II: Explained Variance by Principal Components for Dataset 1 and 2.

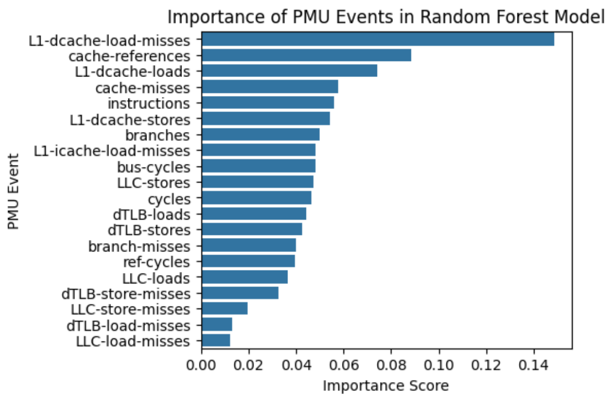
Principal Component	Explained Variance (%)	
	Dataset 1	Dataset 2
PC1	38.3	20.6
PC2	13.3	9.9
PC3	9.3	7.3
PC4	8.3	7.0
PC5	7.0	6.4
PC6	6.4	5.7
PC7	5.0	5.1
PC8	4.4	4.3
Others	8.0	33.7

*2) **Dimensionality Reduction with PCA**:* Given the high dimensionality of the PMU data, which includes 17 and 45 significant events per benchmark for datasets 1 and 2 respectively, Principal Component Analysis (PCA) was employed to efficiently reduce the feature space while preserving the essential variance. The analysis was specifically aimed at retaining at least 90% of the total variance, leading to the reduction of dataset 1 to 8 principal components and dataset 2 to 19 principal components. The first eight principal components of dataset 1 alone accounted for a substantial majority of the data variability, as detailed in Table II, with the first principal component (PC1) capturing 38.3% and 20.6% of the total variance in datasets 1 and 2 respectively. This strategic reduction not only simplifies the complexity of the classification models but also ensures that the analysis retains critical information, emphasizing the robustness of PCA in isolating significant features that drive the machine learning process.

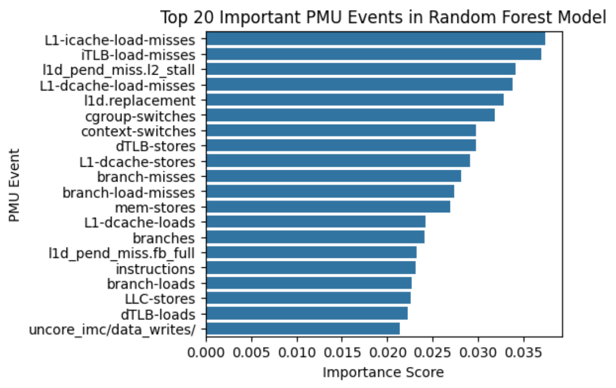
C. Classification Modeling and Results

The classification of benchmark programs based on the extracted PMU event features was performed using several machine learning models, including Logistic Regression, Decision Tree, K-Nearest Neighbors (KNN), and Random Forest. These models were chosen for their ability to capture complex, non-linear relationships in the data. Cross-validation was employed to evaluate the performance of each model, ensuring the robustness and generalizability of the results.

*1) **Evaluation on Dataset 1**:* The preliminary analysis with 10 benchmarks served primarily to test and refine our data processing and machine learning workflows. This smaller dataset allowed for quicker iterations and optimizations, resulting in a

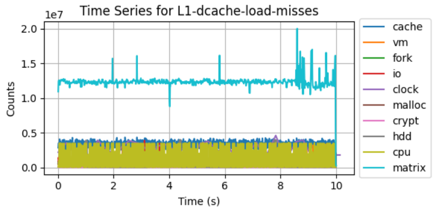


(a) Feature importance from Dataset 1.

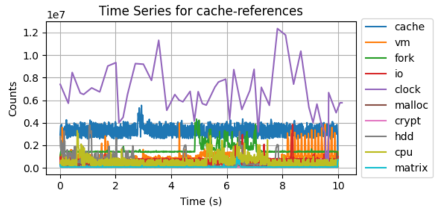


(b) Feature importance from Dataset 2.

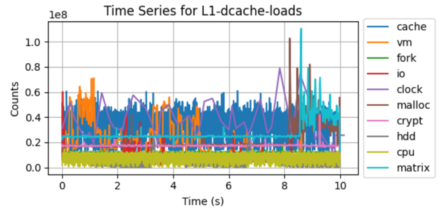
Fig. 4: Feature importance derived from a Random Forest classifier, highlighting the contribution of PMU events to system performance understanding for two datasets.



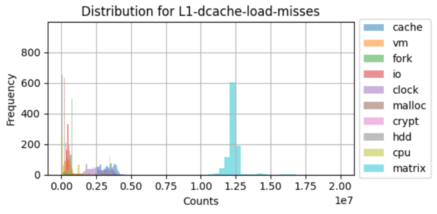
(a) Time-series for *L1-dcache-load-misses*



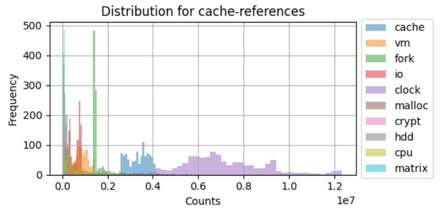
(b) Time-series for *cache-references*



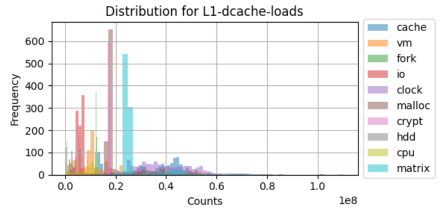
(c) Time-series for *L1-dcache-loads*



(d) Distribution for *L1-dcache-load-misses*



(e) Distribution for *cache-references*



(f) Distribution for *L1-dcache-loads*

Fig. 5: Illustration of PMU events depicting both time-series and distribution plots for selected events and benchmarks from dataset 1. The top row shows time-series data (a-c) and the bottom row shows distribution data (d-f) for each corresponding event, aiding in understanding the dynamic behavior and variability across different benchmarks.

TABLE III: Performance Metrics for Benchmark Classification Models

Dataset	**Classifier**	**Accuracy (%)**	**Precision (%)**	**Recall (%)**	**F1 Score (%)**	**AUC-ROC**
Dataset 1 (10 Benchmarks, 20 PMU events)	Logistic Regression	94.4	94.3	94.4	94.4	0.994
	Decision Tree	98.2	98.2	98.2	98.2	0.990
	K-Nearest Neighbors	98.2	98.2	98.2	98.2	0.997
	Random Forest	99.7	99.7	99.7	99.7	0.99998
Dataset 2 (123 Benchmarks, 81 PMU events)	Logistic Regression	93.3	93.5	93.3	93.3	0.999
	Decision Tree	97.9	97.9	97.9	97.9	0.989
	K-Nearest Neighbors	97.4	97.4	97.4	97.4	0.996
	Random Forest	99.6	99.7	99.6	99.7	0.9997

robust methodology capable of handling more complex data. To assess the effectiveness of PMU data in distinguishing between different benchmarks, we employed several machine learning algorithms, including Logistic Regression (LR), Decision Trees (DT), K-Nearest Neighbors (KNN), and Random Forests (RF). Each algorithm was trained on the principal components derived from the PCA analysis.

The performance of these classifiers, in terms of accuracy, precision, recall, F1 score, and AUC-ROC, is summarized in Table III. The results demonstrate that the Random Forest algorithm achieved the highest accuracy of 99.7% for dataset 1, significantly outperforming the other models. This high level of accuracy indicates that PMU data, when appropriately processed and reduced using PCA, can effectively differentiate between various benchmarks. This finding underscores the potential of PMU data as a viable source for identifying specific system behaviors, which can be leveraged in the context of side-channel attacks.

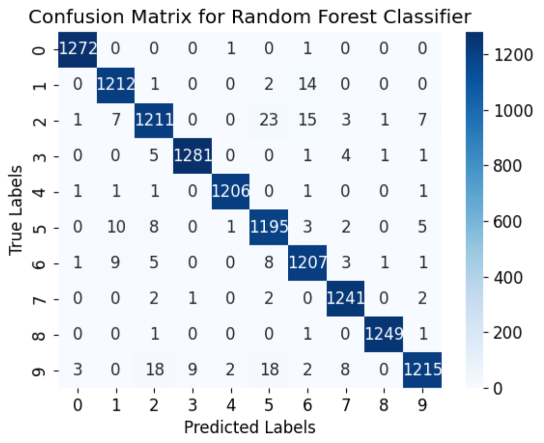


Fig. 6: Confusion matrix of the Random Forest classifier showing the high accuracy of benchmark classification across different classes.

Figure 6 shows the confusion matrix for the Random Forest classifier for dataset 1, highlighting its predictive accuracy across different benchmark classes. The systematic approach to feature analysis and reduction underscores the effectiveness of using PMU data for detailed performance characterization and benchmark classification.

*2) **Evaluation on Dataset 2:*** The expanded analysis included 123 benchmarks, significantly enhancing the complexity and diversity of the dataset. This scale-up was crucial for testing the robustness and generalizability of our machine-learning models across a wider array of system activities. Here, the Random Forest model excelled, demonstrating superior performance with over 99.6% accuracy, suggesting a strong predictive capability across diverse system operations.

The transition from 10 to 123 benchmarks marked significant progress in understanding the relationship between PMU events and system activities. It highlighted specific benchmarks that are particularly indicative of system behavior, aiding in fine-tuning our models for better accuracy and efficiency. The superior performance of the Random Forest classifier, with an AUC-ROC of 0.9997, illustrates its robustness in identifying even subtle differences between benchmarks based on PMU data. This suggests that PMU data, reduced via PCA and classified using machine learning models, holds substantial promise for detecting and mitigating potential side-channel vulnerabilities in modern computing systems.

D. Real-Time Activity Detection Potential

The high classification accuracy achieved with both datasets underscores the feasibility of using PMU data for real-time user activity detection. Such capabilities could be transformative in security contexts, enabling early detection of anomalous behaviors, or in adaptive systems, optimizing resource allocation based on detected activities. However, real-world deployment poses challenges, including handling noise in dynamic environments, adapting to new workloads, and ensuring real-time processing. Future work will focus on addressing these challenges by developing algorithms capable of real-time data analysis and by integrating machine learning models that can adapt to new patterns of activity without requiring frequent retraining. The insights gained from this study pave the way for innovative monitoring solutions that enhance both security and operational efficiency, opening new avenues for research and development in intelligent systems monitoring.

VI. Conclusion and Future Work

This research highlights the security risks associated with exploiting Performance Monitoring Unit (PMU) data for remote side-channel attacks. We demonstrated that microarchitectural fingerprints derived from PMU data could be leveraged to accurately infer user activities in shared system environments, achieving an average prediction accuracy of 98% across multiple machine learning models, including logistic regression, decision trees, k-nearest neighbors, and random forests. These findings reveal latent vulnerabilities in modern computing systems, where PMU data—a tool intended for benign diagnostics and performance optimization—can be misused for covert surveillance. The study underscores the urgency for robust countermeasures to mitigate these emerging threats, such as restricting PMU access, introducing noise injection mechanisms, and implementing hardware-level protections to minimize side-channel leakage.

Future work will focus on expanding the scope of this research to uncover additional vulnerabilities and enhance detection models. Investigating a broader range of PMU events and incorporating advanced machine learning techniques, such as ensemble methods and deep learning, could further refine accuracy and adaptability. Developing real-time detection frameworks is another critical step, enabling instantaneous identification and mitigation of side-channel threats. Additionally, extending this work to multi-tenant cloud environments, where shared hardware amplifies security risks, will provide practical insights into safeguarding infrastructure against PMU exploitation. By advancing the understanding of PMU-based side-channel vulnerabilities and proposing actionable directions for mitigation, this study contributes to the evolving discourse on microarchitectural security and privacy in modern computing systems.

References

[1] Q. Ge, Y. Yarom, D. Cock, and G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, pp. 1–27, 2018.

[2] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[3] R. Azimi, D. K. Tam, L. Soares, and M. Stumm, “Enhancing operating system support for multicore processors by using hardware performance monitoring,” *ACM SIGOPS Operating Systems Review*, vol. 43, no. 2, pp. 56–65, 2009.

[4] L. L. Woo, “Hardware performance counters (hpcs) for anomaly detection,” *Hardware Supply Chain Security: Threat Modelling, Emerging Attacks and Countermeasures*, pp. 147–165, 2021.

[5] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, “Real-time detection for cache side channel attack using performance counter monitor,” *Applied Sciences*, vol. 10, no. 3, p. 984, 2020.

[6] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, “Sok: The challenges, pitfalls, and perils of using hardware performance counters for security,” in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 20–38.

[7] J. Kim, S. van Schaik, D. Genkin, and Y. Yarom, “ileakage: Browser-based timerless speculative execution attacks on apple devices,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 2038–2052.

[8] P. Qiu, Y. Lyu, H. Wang, D. Wang, C. Liu, Q. Gao, C. Wang, R. Sun, and G. Qu, “Pmuspill: The counters in performance monitor unit that leak sgx-protected secrets,” *arXiv preprint arXiv:2207.11689*, 2022.

[9] P. Qiu, Q. Gao, D. Wang, Y. Lyu, C. Wang, C. Liu, R. Sun, and G. Qu, “Pmu-leaker: Performance monitor unit-based realization of cache side-channel attacks,” in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 664–669.

[10] Y. Yang, P. Qiu, C. Wang, Y. Jin, Q. Gao, X. Li, D. Wang, and G. Qu, “Exploration and exploitation of hidden pmu events,” in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[11] Y. Zhang, “Cache-based side-channel attacks in multi-tenant public clouds and their countermeasures,” 2014.

[12] D. A. Osvik, A. Shamir, and E. Tromer, “Cache attacks and countermeasures: the case of aes,” in *Topics in Cryptology–CT-RSA 2006: The Cryptographers’ Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2005. Proceedings*. Springer, 2006, pp. 1–20.

[13] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, “Meltdown,” *arXiv preprint arXiv:1801.01207*, 2018.

[14] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-vm side channels and their use to extract private keys,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 305–316.

[15] G. Contreras and M. Martonosi, “Power prediction for intel xscale® processors using performance monitoring unit events,” in *Proceedings of the 2005 international symposium on Low power electronics and design*, 2005, pp. 221–226.

[16] G. Tsafack Chetsa, L. Lefèvre, J. Pierson, P. Stolf, and G. Da Costa, “Exploiting performance counters to predict and improve energy performance of hpc systems,” *Future Generation Computer Systems*, vol. 36, pp. 287–298, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X13001556

[17] B. Hettwer, S. Gehrer, and T. Güneysu, “Applications of machine learning techniques in side-channel attacks: a survey,” *Journal of Cryptographic Engineering*, vol. 10, no. 2, pp. 135–162, 2020.

[18] J. Gravellier, J.-M. Dutertre, Y. Teglia, P. L. Moundi, and F. Olivier, “Remote side-channel attacks on heterogeneous soc,” in *Smart Card Research and Advanced Applications: 18th International Conference, CARDIS 2019, Prague, Czech Republic, November 11–13, 2019, Revised Selected Papers 18*. Springer, 2020, pp. 109–125.

[19] D. Genkin, N. Nissan, R. Schuster, and E. Tromer, “Lend me your ear: Passive remote physical side channels on {PCs},” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 4437–4454.

[20] Y. Wang, R. Paccagnella, E. T. He, H. Shacham, C. W. Fletcher, and D. Kohlbrenner, “Hertzbleed: Turning power {Side-Channel} attacks into remote timing attacks on x86,” in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[21] G. Sangeetha and G. Sumathi, “An optimistic technique to detect cache based side channel attacks in cloud,” *Peer-to-Peer networking and Applications*, vol. 14, no. 4, pp. 2473–2486, 2021.

[22] S. Banerjee, S. Wei, P. Ramrakhyani, and M. Tiwari, “Triton: Software-defined threat model for secure multi-tenant ml inference accelerators,” in *Proceedings of the 12th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2023, pp. 19–28.

[23] R. B. Gomes, R. D. Medina, and F. G. Moro, “Cloud aid-a cloud computing tool for mitigating side-channel attacks,” in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018, pp. 1–5.

[24] Microsoft, “Tenancy models for a multi-tenant solution,” https://learn.microsoft.com/en-us/azure/architecture/guide/multitenant/considerations/tenancy-models (Accessed on August 26, 2024).

[25] W. J. Brown, V. Anderson, and Q. Tan, “Multitenancy-security risks and countermeasures,” in *2012 15th International Conference on Network-Based Information Systems*. IEEE, 2012, pp. 7–13.

[26] F. Dall, G. De Micheli, T. Eisenbarth, D. Genkin, N. Heninger, A. Moghimi, and Y. Yarom, “Cachequote: Efficiently recovering long-term secrets of sgx epid via cache attacks,” 2018.

[27] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, “A survey of intel sgx and its applications,” *Frontiers of Computer Science*, vol. 15, pp. 1–15, 2021.

[28] A. Kogler, J. Juffinger, L. Giner, L. Gerlach, M. Schwarzl, M. Schwarz, D. Gruss, and S. Mangard, “Collide+Power: Leaking inaccessible data with software-based power side channels,” in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 7285–7302. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/kogler

[29] A. Shusterman, A. Agarwal, S. O’Connell, D. Genkin, Y. Oren, and Y. Yarom, “Prime+Probe 1, JavaScript 0: Overcoming browser-based Side-Channel defenses,” in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 2863–2880. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/shusterman

[30] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss, “Platypus: Software-based power side-channel attacks on x86,” in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 355–371.

[31] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. 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)*, 2018, pp. 7–12.

[32] P. P. Bhade and S. Sinha, “Detection of cache side channel attacks using thread level monitoring of hardware performance counters,” in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2021, pp. 210–217.

[33] U. Mandal, S. Bhattacharya, and D. Mukhopadhyay, “Cache wars: A comparative study of umwait, umonitor, and prime-probe attacks,” in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 86–89.

[34] A. Spence and S. Bangay, “Security beyond cybersecurity: side-channel attacks against non-cyber systems and their countermeasures,” *International Journal of Information Security*, vol. 21, no. 3, pp. 437–453, 2022.

[35] C. I. K. Gavin, “stress-ng (stress next generation),” https://github.com/ColinIanKing/stress-ng (Accessed on August 18, 2024).

[36] I. Corporation, “Intel 64 and ia-32 architectures software developer’s manual,” https://software.intel.com/content/www/us/en/develop/articles/intel-sdm.html (Accessed on August 18, 2024), 2016.

[37] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, “Accurate online power estimation and automatic battery behavior based power model generation for smartphones,” in *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2010, pp. 105–114.

[38] I. T. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.

[39] Scikit-learn, “Feature extraction,” https://scikit-learn.org/stable/modules/feature_extraction.html (Accessed on August 18, 2024).

[40] DataCamp, “Classification in machine learning: An introduction,” https://www.datacamp.com/blog/classification-machine-learning (Accessed on August 18, 2024).

[41] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.