A Real Time Cache Side Channel Attack Detection and Mitigation Framework based on Machine Learning

Youssra Ghabbara
Multimedia, Information Systems
and Advanced Computing Laboratory (MIRACL),
University of Sfax
The Higher Institute of Management of Gabes (ISGG),
University of Gabes, Tunisia
e-mail: ghabbarayoussra@gmail.com

Zied Trifa
Multimedia, Information Systems
and Advanced Computing Laboratory (MIRACL),
University of Sfax
The Higher Institute of Management of Gabes (ISGG),
University of Gabes, Tunisia
e-mail: trifa.zied@gmail.com

***Abstract*—Cloud computing has transformed data management by offering scalable and flexible services to organizations. However, its multi-tenant architecture, where users share physical hardware, introduces critical security risks related to data isolation and resource sharing. Among these, cache side-channel attacks (CSCAs) represent a particularly dangerous threat. These attacks exploit shared cache memory, using variations in cache access times to extract sensitive information from co-located virtual machines. The shared nature of cloud hardware and the difficulty of detecting such attacks in real-time make cache side-channel attacks especially challenging to address. Traditional security measures, such as encryption and access control, fall short against these threats, as they do not target vulnerabilities in the cloud underlying hardware architecture. This research proposes a real-time detection and mitigation framework leveraging machine learning to address the pressing issue of cache side-channel attacks. Key focuses of this study include designing an efficient feature extraction mechanism to identify malicious cache behaviors and selecting machine learning algorithms that provide high detection accuracy with minimal latency. Additionally, the framework incorporates real-time mitigation strategies designed to minimize performance degradation in cloud environments.**

***Keywords*-Cache side-channel attack; Hardware performance counters; Machine learning; Cloud Computing.**

I. Introduction

The rapid development of information technology has significantly impacted industries, leading to the adoption of cloud computing. However, this technology presents security challenges, particularly in data segregation and resource allocation [1]. Current detection systems struggle to distinguish legitimate and malicious activity, leading to false alarms and stealth attacks. An effective detection system must balance real-time responsiveness with minimal system performance, but prior frameworks such as [2][3], often lack precision or require high resource consumption.

To address these challenges, this research aims to develop a real-time detection and mitigation framework for cache side-channel attacks in cloud environments. It explores how machine learning can improve detection accuracy while keeping system overhead low, and identifies mitigation strategies that counter threats with minimal impact on cloud performance. The framework integrates a hybrid model combining Random Forest and XGBoost with an intelligent noise injection mechanism to reduce performance degradation, achieving a balance between security, accuracy, and efficiency for real-time applications.

- Hardware Performance Counters (HPCs) [4] Data: This study uses hardware performance counters to collect features associated with Cache Side-Channel-Attacks (CSCAs) under various attack scenarios and load conditions. Using Greedy Forward Selection and Pearson Correlation in Waikato Environment for Knowledge Analysis (WEKA)[5], we identified an optimal subset of features to inform the detection model.

- Unified Detection Model: Unlike previous studies that analyze attacks in isolation, this paper establishes a unified detection model for multiple CSCAs, including Flush+Reload (FR) [6], Flush+Flush (FF) [7], Prime+Probe (PP) [8], and Spectre [9], achieving high detection accuracy under different system loads.

The structure of this paper is as follows: Section 2 provides a detailed analysis of cache side-channel attacks and introduces hardware performance counters, along with related work. Section 3 describes the machine learning model's setup. Section 4 discusses the experimental framework, including the selection of optimal features and machine learning algorithms. Section 5 presents results across different system loads and evaluates model performance. Finally, Section 6 offers a summary and discussion of the research findings.

II. Related work

Mushtaq et al. [2] analyze FF, FR, and PP attacks by counting the different characteristics of the hardware counters. The highest detection accuracy can reach 99.51% however, 12 different hardware performance events are used for modeling, which is significantly different from the actual 4 hardware counter interfaces and cannot be applied to real-time monitoring of cache attacks. In 2016, Zhang et al. [3] established a detection model by analyzing side-channel attacks based on cross-virtual machine caching, namely FR and PP. The detection method they proposed is to correlate the execution of encrypted applications with the cache miss/hit rate of untrusted virtual machines, but they did not consider whether it would affect their detection success rate under different system loads; moreover, only using the ratio of hits and misses of cache as an indicator can provide too little information, and it is

easy to misjudge some benign programs. S. Mahipal et al. [10] introduced an innovative solution to detect and mitigate cache side channel attacks within virtualized environments; their approach leverages a softmax function grounded in machine learning principles, complemented by an intelligent noise addition technique for effective mitigation. This framework relies primarily on central processing unit (CPU) counters to function optimally; however, it is important to note that such counters may not be universally accessible across all virtualized environments. Li et al. [11] propose an online detection of Spectre attack by monitoring microarchitectural features using time series classification, however it only targets Spectre attack and is not able to provide more comprehensive protection from Side Channel Attacks. Allaf et al. [12] developed machine learning models to identify attacks on FR and PP. The detection success rate under no-load conditions can reach 97%, but under load conditions, the detection success rate drops significantly, even lower than 70%. Moreover, the above papers model each type of attack separately, which loses the significance of detection as the first door of protection in the actual application process. The advantage of this paper is that, using as few hardware events as possible to establish a unified model, the final model can accurately find out whether there is any one of the four cache side-channel attacks.

III. Methodology

A. Cache side attacks

Current processors run at high clock speeds, but their performance is hindered by slow main memory access times, leading to a bottleneck. High-speed caches, situated between the processor and main memory, mitigate this issue by offering a faster memory bandwidth. Modern cache structures include three levels: Last-Level-Cache (LLC) (level 3), L2 (level 2), and L1 (level 1, with L1 Data and L1 Instruction). While enhancing efficiency, caches also present security risks, particularly through techniques like Flush+Flush, Flush+Reload, Spectre, and Prime+Probe, which exploit access time differences [13]. The described techniques exploit cache timing discrepancies to extract sensitive information in virtualized environments. The Prime+Probe [8] method allows attackers to gauge cache states by measuring access times after filling the cache, indicating whether the victim has accessed a specific cache line. In contrast, Flush+Reload and Flush+Flush enhances [6][7] stealth by relying solely on timing observations without memory accesses, signaling victim interaction through execution time increases. Specter attacks [9] exploit Central Processing Unit (CPU) mispredictions to manipulate branch prediction, enabling attackers to access otherwise restricted data and recover them through side-channel methods.

B. Hardware Performance Counters (HPCs)

HPCs are system-specific registers built into x86 and Advanced RISC Machines(ARM) processors, originally designed for software debugging and system performance analysis. They now serve as a tool for detecting security risks and program vulnerabilities, as they can detect features during program execution, reducing performance overhead and making them ideal for cache side-channel attacks.

C. Experimental Setup and Data Collection

All tests in this study were carried out on an 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40 GHz desktop computer equipped with 8 cores, 8GB RAM, and a three-level cache system. The victim applications and side-channel attacks were selected from Mastik [14] and Xlate [15]. Moreover, the MiBench [16] benchmark suite was employed to represent benign applications. In this study, we utilized the Perf tool to collect hardware performance counters from Model-Specific Registers (MSRs). The recommended tool collects HPCs by processor in microseconds with exclusive access to avoid contamination from other processes, tackling the issues of overcounting noted in a recent study [4]. This study considers 14 HPCs features from the table I for further analysis.

TABLE I
THE COLLECTED HPC FEATURES AND THEIR RANKING

Ranking	HPC Name	Ranking	HPC Name
1	cache-references	8	L1-icache-load-misses
2	cache-misses	9	LLC-misses
3	CPU-cycles	10	iTLB-load-missses
4	instructions	11	LLC-store-misses
5	branches	12	LLC-loads
6	branch-misses	13	dTLB-load-misses
7	L1-dcache-load-misses	14	branch-instructions

HPCs data is collected using the four available HPCs registers of the tested I5 processor, with readings taken at 50-microsecond intervals. Each instance of a Victim under No Attack (VNA) and a Victim under Attack (VA) is executed 50 times under two distinct load conditions: No Load (NL) and Full Load (FL). The "NL" scenario refers to a system solely processing victim applications, while the "FL" scenario involves the system handling victim applications on one core while benign applications are executed on the remaining cores. Afterward, the HPCs data from both VA and VNA executions is consolidated to create the final dataset for analysis.

D. Customized Features based Classifier

The proposed classification based on customized features consists of two main steps as shown in Figure 1: 1) feature selection and 2) features reduction due to a limited number of registers for effective real-time detection of attacks.

For feature selection, the raw data will be transformed from a time-based subsequence into a feature vector for classification purposes. To identify the most relevant features for attack detection, we applied the Greedy Forward Selection algorithm [17][18]. This method starts with an empty set of features and incrementally evaluates the impact of adding each feature. At each iteration, the algorithm incorporates the feature that results in the greatest improvement in model performance, as measured by metrics such as accuracy.

For feature reduction, detecting cache side-channel attacks using machine learning requires low-level feature representation, resulting in high-dimensional data that complicates processing

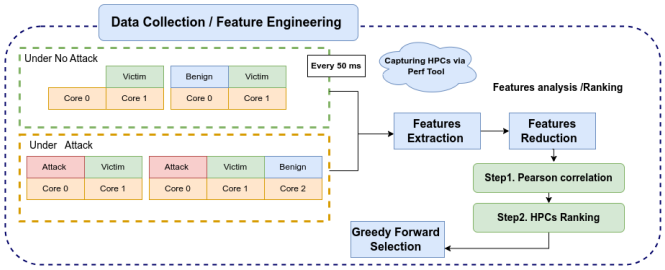

Figure 1. Data Collection and Features Engineering.

and increases computational overhead. An excessive number of features can negatively affect the accuracy and performance of the classifier, making effective feature reduction essential [19]. Our objective is to identify a limited set of significant HPCs that can be gathered in real-time from low-end processors with minimal disruption. We utilized the Correlation Attribute Evaluation method in WEKA to compute Pearson correlation coefficients [20], retaining only features that exceed a specified correlation threshold (e.g., 0.4).

The Pearson correlation coefficient (adapted from [21]) between each attribute and class is calculated, as given below:

$$\rho(i) = \frac{\text{cov}(X_i, Y)}{\sqrt{\text{var}(X_i) \cdot \text{var}(Y)}} \quad (1)$$

where:

ρ is the Pearson correlation coefficient.

X_i is the input dataset of event i (where $i = 1, \ldots, 14$).

Y is the output dataset containing labels, i.e., “attack” or “no attack” in this case.

$\text{cov}(X_i, Y)$ measures the covariance between input data and output data.

$\text{var}(X_i)$ and $\text{var}(Y)$ measure the variance of both input and output datasets, respectively.

The sum score of each HPC feature will be calculated and HPCs will be ranked according to sum score as shown in Table 1.

E. ML Classifiers

1) Random Forest (RF): RF [22] is a widely used machine learning classification model consisting of Decision Trees (DTs) based on specific parameters. DTs struggle with data fluctuations, but RF incorporates independent DTs with randomly generated datasets. Low correlation between trees improves output accuracy, as high correlation can lead to incorrect decisions. The majority of trees decide the final outcome, with their vote determining it.

2) Extreme Gradient Boosting (XGBoost): XGBoost represents an ensemble learning technique founded on gradient boost, frequently used by researchers to predict distributed denial-of-service (DDoS) attacks. XGBoost combines a linear model with a boost tree model, leveraging not only the first derivative, but also the second derivative of the loss function for second-order differentiation. This strategy speeds up convergence to global optimality, improving the overall efficiency of the model’s solution [23].

After experimenting with RF and XGBoost algorithms for CSCAs detection, it was apparent that the obtained results were not satisfactory. As a result, we opted for an ensemble approach to develop a detection system for CSCAs based on an ensemble learning technique. This technique involves combining multiple machine-learning models to improve the overall accuracy of the detection system. This model consisted of two machine learning algorithms, namely RF and XGBoost. The voting mechanism used was ’Soft’, where the final prediction is determined by the average vote of the constituent models.

F. Attack mitigation

An algorithm known as Intelligent Noise Addition for Attack Mitigation (INA-AM) (which was first introduced and adapted from [10]), is defined to mitigate the effect of different kinds of side-channel attacks. As depicted in Figure 2, it takes cache hits and cache misses vectors as inputs and gives noisy cache hits and noisy cache misses as outputs to confuse attackers and mitigate cache side-channel vulnerabilities.

Algorithm: Intelligent Noise Addition for Attack Mitigation (INA-AM)
Input: Cache Hits H, Cache Misses M
Output: Noise Cache Hits H’, Noise Cache Misses M’

1. Start
2. Initialize noisy cache hits vector H’
3. Initialize noisy cache misses vector M’
4. hcount ← CountCacheHits(H)
5. mcount ← CountCacheMisses(M)
6. noise function ← ComputeNoiseFunction(H, M)
7. For each cache hit h in H
8. IF the noise function recommends noise to h Then
9. Add noise to h
10. Add h to H’
11. End If
12. End For
13. For each cache miss m in M
14. IF the noise function recommends noise to m Then
15. Add noise to m
16. Add h to M’
17. End If
18. End For
19. Output H’
20. Output M’
21. End

Figure 2. Intelligent Noise Addition for Attack Mitigation (INA-AM).

Vectors H’ and M’ are initialized to store noisy cache hits and misses in Step 2 and Step 3 having a smart noise function is crucial in order to confuse the attacker, who relies on distinguishing between low latency cache hits and high latency cache misses. In Step 4 and Step 5, the number of cache hits and cache misses are calculated. In Step 6, the noise function is calculated to assist in adding necessary noise. The noise function is used for an iterative process from Step 7 to Step 12 to introduce noise into cache hits. Similarly, a different iterative method is employed to introduce noise to cache misses ranging from Step 13 to Step 18. In conclusion, the algorithm generates noisy hits and misses vectors to prevent successful attacks.

IV. Results

A. Ensemble Model Evaluation

As previously discussed in this paper, we employed a hybrid model that integrates both RF and XGBoost classifiers. To thoroughly assess the performance of our model, we first evaluated each classifier independently. Figure 3 illustrates the accuracy and false alarm rate achieved by the two classifiers, providing a clear comparison of their individual performance metrics.

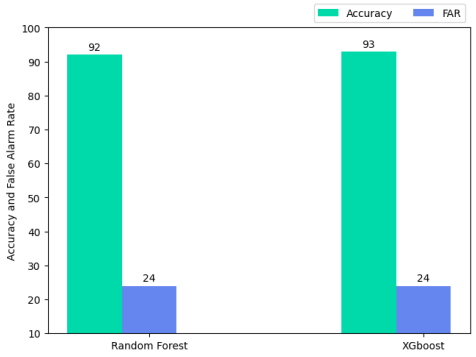

Figure 3. Accuracy and FAR for RF and XGBoost.

The bar chart illustrates the accuracy and False Alarm Rate (FAR) of RF and XGBoost in side-channel attack detection. Random Forest achieves 92% accuracy, while XGBoost reaches 93%. Both models exhibit a significant false alarm rate of 24%, leading to misclassification of benign processes as attacks. This high FAR can cause disturbances and poor performance, highlighting the need for better detection reliability while minimizing false alarms.

In Figure 4 we will see the comparison between the two classifiers and the hybrid model.

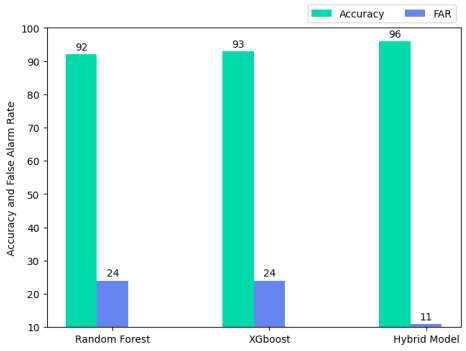

Figure 4. Comparison of Accuracy and FAR for the three Models.

RF achieves an accuracy of 92% with a FAR of 24%, while XGBoost improves slightly with the accuracy of 93%, but maintains the same FAR of 24%. In contrast, the hybrid model significantly outperforms both individual models, achieving the highest accuracy of 96% and drastically reducing the FAR to 11%. This indicates that the hybrid model not only enhances detection accuracy but also substantially reduces false positives, making it a more effective and reliable solution for real-time CSCAs detection in cloud environments.

B. Customized Features vs Normal Features

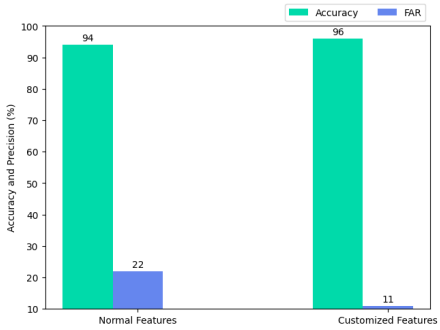

Figure 5. Comparison of Customized Features and Normal Features of the Model.

The graph in Figure 5 illustrates the clear benefits of customized features over normal features in detecting CSCAs. The model with customized features achieves 96% accuracy, surpassing the 94% of the normal features model. Though the accuracy difference is minimal, it highlights the importance of customized features in enhancing precision, particularly for real-time attack detection. Additionally, false alarm rates decrease significantly from 22% to 11%, demonstrating the effectiveness of customized features in improving detection reliability and overall system efficiency.

After training a machine learning model, the next step is to assess how effective the model is. There exist various metrics which can be used to evaluate a machine learning-based trained model for binary and multi-class classification problems. These metrics include precision, precision, True Positive Rate (TPR), False Positive Rate (FPR), F1 score, Area Under Curve (AUC), and Receiver Operating Characteristic (ROC).

The performance indicators for this model are shown in Table II.

TABLE II
Performance Evaluation Metrics

Evaluation Metrics	Environment	
	NL	FL
Precision	0.98	0.98
F1-score	0.99	0.98
True Positive Rate (TPR)	1.00	0.98
False Alarm Rate (FAR)	0.11	0.11

Figures 6 and 7 below show the ROC curve of the ensemble model, illustrating their ability to diagnose four types of attacks under variable load conditions.

The ROC curve illustrates the true positive and false positive rates in the ensemble classification model. The AUC metric evaluates model performance, with a higher AUC indicating better classification. The model's AUC values are 0.95 for NL and 0.94 for FL, both nearing 1 under various loads, reflecting

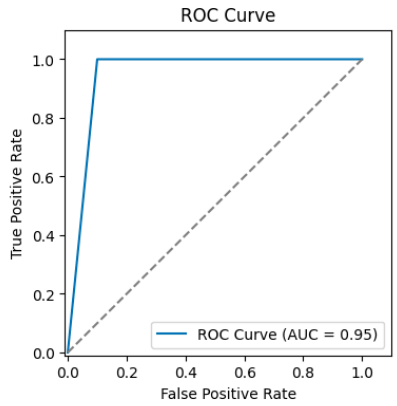

Figure 6. ROC Curves for Model under NL.

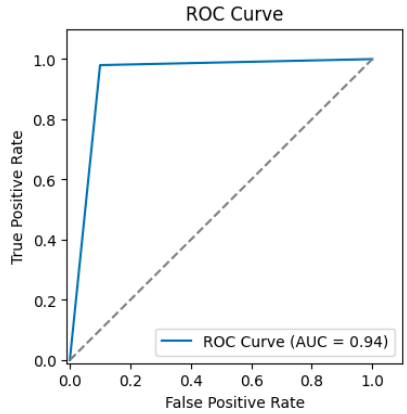

Figure 7. ROC Curves for Model under FL.

excellent classification performance. Thus, the detection model is effective both in actual load conditions and in no load conditions.

To further evaluate the effectiveness of our proposed detection framework, we conducted tests against stealth attacks, which are known for their ability to evade traditional detection mechanisms. As shown in Table III, our framework

TABLE III
INFORMATION LEAKAGE AGAINST STEALTH ATTACK

Attack Detection Method	Information Leakage due to one Stealth Attack (bits)
ICM in [24]	246
Method on AES Encryption in [25]	345
HPC method in [2]	195
Proposed Detection Model	0

demonstrated superior resilience to these attacks compared to several state-of-the-art methods. Although other detection techniques, such as the Unsupervised Deep Learning (UDL) [26], Intel Cache Monitoring Technology (ICM) [24], Hardware Performance Counters (HPC)-based models [2] and AES Encryption in [25], exhibited significant levels of information leakage during a single stealth attack, ranging from 189 to 345 bits. our approach effectively prevented information leakage achieving a measured leakage of zero bits.

C. Comparison with the state of art

The proposed attack detection method has been thoroughly compared with advanced techniques, including Unsupervised Deep Learning (UDL) [26], Intel Cache Monitoring Technology (ICM) [24], Hardware Performance Counters (HPC), AES encryption-based methods [25], and standard HPC approaches [2]. The analysis emphasizes data generation tools, performance counters, detectable cache side-channel attacks, and the ability to identify stealth attacks, showcasing the advantages of the proposed framework.

Table IV illustrates various methods employing distinct tools and performance counters for attack detection. The UDL method utilized the Intel Performance Counter Monitor (PCM), with l1 cache hits and misses as performance counters, successfully identifying flush-reload and prime-probe attacks. Similarly, the ICM and HPC methods employed the CMT tool to detect the same attacks using l1 and llc cache misses. An AES encryption method also used Cache Monitoring Technology (CMT) for the dataset. The HPC method adopted the Linux perf tool but focused solely on llc cache misses for attack detection.

TABLE IV
PERFORMANCE COMPARISON AMONG DIFFERENT SIDE-CHANNEL ATTACK DETECTION MODELS

Attack Detection Method	**The tool used to Generate the Dataset**	**Used Performance Counters**	**Cache Side-Channel Attacks Detected**	**Ability to Detect Stealth Attack?**
UDL method in [26]	PCM tool from Intel	L1-INST-MISS L1-INST-HIT LLC-MISS	Flush-Reload Prime-Probe	N
ICM and HPC method in [24]	CMT tool from Intel	L1-MISS LLC-MISS	Flush-Reload Prime-Probe	N
Method on AES Encryption in [25]	CMT tool from Intel	L1-MISS LLC-MISS	Flush-Reload Prime-Probe	N
HPC method in [2]	perf tool in Linux	LLC-MISS	Flush-Reload Prime-Probe	N
Proposed Detection Model	**perf tool in Linux**	**LLC-MISS instructions branches branch-instructions**	**Flush-Reload Prime-Probe Flush-Flush Spectre**	**Y**

In contrast, the proposed method utilizes the perf tool, incorporating four performance counters and machine learning, enabling it to detect all attack types, including stealth.

V. Conclusion and Future Work

As cloud computing expands, security challenges such as CSCAs in multi-tenant environments have become a growing concern. These attacks exploit microarchitectural vulnerabilities, often bypassing traditional security measures. This research introduced a machine learning-based framework for real-time detection and mitigation, combining RF and XGBoost for improved accuracy and employing an intelligent noise injection mechanism to minimize data leakage with minimal performance overhead. While the framework demonstrated promising results, challenges remain. The FAR highlights the need for further model optimization, and its scalability across different cloud architectures requires further evaluation. Future research will focus on integrating deep learning to enhance detection capabilities, developing adaptive mechanisms for dynamic threat response, and conducting large-scale testing to ensure real-world applicability. Additionally, strategies for managing software updates and long-term system maintenance will be explored. Addressing these challenges is key to strengthening cloud security and advancing real-time attack mitigation.

References

[1] M. S. Inci, B. Gülmezoglu, G. I. Apecechea, T. Eisenbarth, and B. Sunar, "Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud", *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 898, 2015.

[2] M. Mushtaq *et al.*, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters", in *Proc. 7th Int. Workshop Hardware Archit. Support Security Privacy*, 2018.

[3] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds", in *Proc. Int. Symp. Recent Advances Intrusion Detection*, 2016.

[4] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks", *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 564, 2017.

[5] I. H. Witten and E. Frank, "Data mining: Practical machine learning tools and techniques", in *The Morgan Kaufmann Ser. Data Manag. Syst.*, 2005.

[6] Y. Yarom and K. E. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack", *IACR Cryptol. ePrint Arch.*, vol. 2013, p. 448, 2014.

[7] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack", in *Proc. Int. Conf. Detection Intrusions Malware Vulnerability Assessment*, 2015.

[8] G. I. Apecechea, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes", in *Proc. IEEE Symp. Security Privacy*, 2015, pp. 591–604.

[9] P. C. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution", in *Proc. IEEE Symp. Security Privacy (SP)*, 2019, pp. 1–19.

[10] S. Mahipal and V. C. Sharmila, "A security framework for improving qos by detecting and mitigating cache side-channel attacks in virtualized environments", 2023.

[11] C. Li and J.-L. Gaudiot, "Online detection of spectre attacks using microarchitectural traces from performance counters", in *Proc. 30th Int. Symp. Comput. Archit. High Perform. Comput. (SBAC-PAD)*, 2018, pp. 25–28.

[12] Z. Allaf, M. Adda, and A. E. Gegov, "A comparison study on flush+reload and prime+probe attacks on aes using machine learning approaches", in *Proc. UK Workshop Comput. Intell.*, 2017.

[13] D. Page, "Theoretical use of cache memory as a cryptanalytic side-channel", *IACR Cryptol. ePrint Arch.*, vol. 2002, p. 169, 2002.

[14] Y. Yarom, *Mastik: A micro-architectural side-channel toolkit*, 2016.

[15] M. Chiappetta, E. Savas, and C. Yilmaz, *Xlate*, 2016.

[16] M. R. Guthaus *et al.*, "Mibench: A free, commercially representative embedded benchmark suite", in *Proc. 4th Annu. IEEE Int. Workshop Workload Characterization (WWC-4)*, 2001, pp. 3–14.

[17] T. Pahikkala, A. Airola, and T. Salakoski, "Speeding up greedy forward selection for regularized least-squares", in *Proc. 9th Int. Conf. Mach. Learn. Appl.*, 2010, pp. 325–330.

[18] X.-T. Yuan and S. Yan, "Forward basis selection for sparse approximation over dictionary", in *Proc. Int. Conf. Artif. Intell. Statist.*, 2012.

[19] H. Sayadi, N. Patel, A. Sasan, and H. Homayoun, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures", in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, 2017, pp. 129–136.

[20] G. D. Kader and C. Franklin, "The evolution of pearson's correlation coefficient", *Math. Teacher: Learn. Teach. PK–12*, vol. 102, pp. 292–299, 2008.

[21] H. Wang, H. Sayadi, S. Rafatirad, A. Sasan, and H. Homayoun, "Scarf: Detecting side-channel attacks at real-time using low-level hardware features", in *Proc. 26th Int. Symp. On-Line Testing Robust Syst. Design (IOLTS)*, 2020, pp. 1–6.

[22] L. Breiman, "Random forests", *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[23] D. Van, *Ensemble methods: Foundations and algorithms*, 2012.

[24] M. Mushtaq *et al.*, "Run-time detection of prime + probe side-channel attack on aes encryption algorithm", in *Proc. Global Inf. Infrastruct. Netw. Symp. (GIIS)*, 2018, pp. 1–5.

[25] N. H. Ali, M. E. Abdulmunem, and A. E. Ali, "Learning evolution: A survey", *Iraqi J. Sci.*, 2021.

[26] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Südholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters", in *Proc. 3rd Int. Conf. Fog Mobile Edge Comput. (FMEC)*, 2018, pp. 7–12.