Multi Hardware-Attack Dataset and ML-based Detection Using Processor Stress Patterns on x86

1st David Andreu
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain
david.andreu.gerique@estudiantat.upc.edu

2nd Beatriz Otero
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain
beatriz.otero@upc.edu,

3rd Ramon Canal
Department of Computer Architecture
Universitat Politècnica of Catalunya
Barcelona, Spain
ramon.canal@upc.edu,

***Abstract*—Hardware attacks exploit the vulnerabilities discovered in state-of-the-art CPUs. As an example, attacks such as Meltdown and Spectre have made the headlines. To benefit from the vulnerabilities, hardware attacks stress tremendously some section/s of the processor, usually the branch-prediction unit and the different cache levels. This gives us a recognizable pattern and a way to implement a system capable of detecting the presence of these attacks while monitoring the computer. In this paper, we describe the set of hardware attacks under focus, then we describe how we create the dataset and, finally, the use of machine learning to detect the attacks in three scenarios (i.e. training on both benign applications and attacks, training on only benign applications and training only on attacks) and two x86 CPUs (Intel and AMD). The techniques proposed are capable of achieving over 99% detection rate with a machine learning model. This provides a run-time solution to quickly identify the attack as it starts running and take remedial actions.**

***Index Terms*—Security, hardware attack, Spectre, Meltdown, Fallout, machine learning**

I. Introduction

In today's world, computers are integral to our daily lives, from work desktops to personal smartphones. We trust these devices to securely store our data, but this trust is often misplaced, as we are continually at risk of cyber attacks. Most modern attacks exploit vulnerabilities in operating systems, with *privilege escalation* being the most common goal. These software-based vulnerabilities can often be quickly fixed through updates. In contrast, *hardware-based attacks*, like Meltdown [1] and Spectre [2], target vulnerabilities in the microprocessor itself. These hardware attacks typically stress parts of the processor, such as the branch-prediction unit and caches, creating recognizable patterns that can uncover them. Modern solutions such as KPTI [3] are effective in mitigating many hardware attacks. However, the objective is to develop a solution that can also address future attacks. By employing machine learning's pattern recognition capabilities, similar behaviours in new exploits can be identified. The objective of this work is to develop, train, and fine-tune a machine learning (ML) model capable of detecting both current and future hardware attacks by learning their characteristic patterns.

Specifically, in this work, we will refer to hardware attacks as a set of attacks falling under the category of hardware-based attacks known as cache side-channel attacks. Cache side-channel attacks are a type of attack that exploit unintentional information leaks within the processor's cache system. These attacks use variations in cache access times, storage patterns, or eviction behaviours to infer sensitive information such as cryptographic keys or private user data. These attacks leverage the subtle changes in how data is stored, accessed, or evicted across different cache levels, enabling attackers to deduce the operations performed by a program without directly accessing the target data. This poses a significant and sophisticated security threat to modern computing systems.

There are various types of attacks within this category. In this work, we focus on the following cache side-channel attacks: Spectre V1, Spectre V2, Spectre V4, Meltdown, ZombieLoad, Fallout and Crosstalk.

II. Hardware attacks in x86

A. Threat model

This work assumes an *unprivileged attacker* (i.e., without kernel-level access) and the *absence of kernel-level mitigations* against microarchitectural vulnerabilities such as Kernel Address Space Layout Randomization (KASLR) (e.g. KAISER [3] LAZARUS [4], or FLARE [5]) —which are designed for x86 architectures and rely on platform-specific features —are not considered active. This assumption is realistic as we are considering a platform-independent mechanism to detect the attacks.

B. Related Work

Hardware attacks create distinctive performance anomalies, such as an unusual frequency of branch mispredictions or excessive cache evictions. These patterns are often consistent across different executions of the attack, enabling ML models to generalize and detect them reliably. Previous approaches to detecting hardware attacks using hardware performance counters and ML models have shown great performance [6], [7], [8], [9]. However, these efforts focus only on a single attack and do not share their training datasets, models, or instructions for generating similar models locally. This lack of extensive analysis and reproducibility in the area has brought some authors [10] to believe that ML is not appropriate, even disregarding previous work and evidence.

Other studies, like Carnà et al. [7], provide examples of binaries used in attacks, they lack details on data recording

Manuscript received May 19, 2025; revised July 30, 2025;
accepted August 13, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Regular
DOI: doi.org/10.64552/wipiec.v11i1.94

conditions and methods, as well as specifics on benign programs, making exact replication difficult.

The primary goal of this work is to build and release [11] a comprehensive dataset comprising multiple known attacks and describing the creation methodology. With this, we develop a novel ML model for detecting hardware attacks using hardware performance counters (HPCs). This effort aims to facilitate replication and foster further research within the community.

The ML model is designed to enable rapid, real-time analysis of HPC data streams, making it possible to scale the detection mechanism across multiple systems without sacrificing responsiveness or reliability. In summary, our work makes the following contributions:

- **Build a reliable and reproducible dataset**. The dataset must include relevant samples from both hardware attacks and benign programs, correctly labeled and compatible across different machines and architectures. It should be reproducible under the same conditions on other machines, involving:
  - Identify hardware attack binaries and benign programs for data collection.
  - Record HPC data and sample rate.
- **Develop a ML model**. The model should classify input samples as either malicious or benign, and also distinguish between known attacks and benign programs. The ML model will be broken down in the following steps:
  - Identify the optimal ML model for the task.
  - Preprocess data for the selected model and training it.
  - Optimize parameters.

III. ML Models

Machine learning is preferred over deep learning because the dataset is too small. Deep learning models require hundreds of thousands to millions of samples, while the current dataset only has 28,000 samples from 14 different programs. This size is insufficient for deep learning, making traditional ML methods more suitable and recommended for smaller datasets.

This paper uses Naive Bayes, Decision Tree, Random Forest, and Support Vector Machines classifiers for multi-class classification tasks. Each method is briefly described below.

Naive Bayes is a simple and efficient probabilistic classifier based on Bayes' theorem, assuming feature independence [12].

Decision trees are widely used supervised learning algorithms for classification [13]. They have a hierarchical tree-like structure, where the internal nodes represent decisions based on the feature values, the branches represent the decision results, and the terminal nodes represent the classification categories.

Random Forest (RF) is a classification algorithm that builds multiple independent decision trees using bootstrap sampling of the training data [14]. For each split, it randomly selects a subset of features. In classification, each tree votes on the class, and the majority vote determines the final prediction.

Support Vector Machine (SVM) is a supervised learning algorithm for classification tasks, such as distinguishing between benign and malicious samples and identifying specific attacks [15]. SVM uses support vectors, the critical data points closest to the decision boundary (hyperplane), to maximize the margin between classes, improving generalization to new data. The margin can be determined using linear or non-linear functions like polynomial or radial basis functions (RBF).

One-Class SVM, a variant of SVM, is used for anomaly detection by learning the majority class space and identifying deviations as anomalies. This is useful for detecting cyberattacks in datasets with mostly benign or few attack samples by training the model on benign patterns and identifying deviations as potential attacks. This method will be applied to unbalanced datasets where the samples are entirely benign or malicious.

IV. System setup and dataset creation

The platforms selected are two x86 CPUs from different manufacturers: Intel i5-8250U and an AMD Ryzen 7 3700X. In our analysis, some older architectures are not vulnerable to certain attack types. This ensures the ability to create a consistent dataset for each platform with multiple attacks. The hardware attacks selected run successfully in the host machines (thus, we guarantee we log "real" traces). We use Lesimple's **spectre-meltdown-checker** script available on GitHub [16] for this purpose. This script analyzes computer characteristics and available mitigations and provides a list of successful hardware-based attacks on the computer. Section IV-C describes the attacks in detail.

The selection of benign programs is performed to ensure reliable and reproducible execution behavior that mirrors common workloads. Various benchmarks with different focuses will be chosen to maximize dataset coverage. Section IV-D describes the attacks in detail.

A. Selection of HPCs

The selected HPCs should accurately represent the patterns exploited by hardware-based attacks, enabling the detection of anomalies when compared to benign executions. The selected counters must also be generic enough to avoid dependence on specific architectures, ensuring the solution's portability across a wide range of computers. Experimentally, we found that there is a soft limit of four counters before some samples are lost in our system. Therefore, monitoring will be limited to four counters.

Some hardware attacks, like those in the Spectre family, exploit speculative execution, triggered when the branch predictor predicts the outcome of a branch instruction. Both `branch instructions` and `branch misses` are generic *perf* events, providing the ratio between the total branches and those where the predictor missed. This selection is supported by previous work, such as Congmiago Li et al. [6]. Additionally, many hardware-based attacks use side channels to extract information, which heavily stress the computer's cache memory. A high count of cache misses on

the last-level cache (LLC) memory may indicate the presence of a FLUSH+RELOAD side-channel attack, known as the most effective and popular among hardware-based attacks. The first-level cache is also a common target in other attacks, as used by Stefano Carnà et al. [7]. Thus, the other two HPCs to be analyzed will be `LLC-load-misses` and `L1-dcache-load-misses`.

For dataset generation, the *perf* tool will be utilized. This tool enables recording of multiple HPCs during binary execution. The `perf stat` command will output HPC counts in a csv file format. To streamline operations, only one CPU core will be utilized, achieved using the `taskset` Linux tool, ensuring collected HPC data remains unaffected by workload distribution across cores.

B. Sample rate

Another decision to make is the sampling rate. Previous works have used sampling rates ranging from 1 ms per sample to 100 ms per sample. Congmiago Li et al. [6] even dynamically change the sample rate to prevent evasive malware. To generate a large number of samples for the ML model, the aim was to use the lowest possible sample rate. However, experimentally it was found that anything under 10 ms caused anomalies in *perf*, such as some samples not being recorded. Therefore, a 10 ms sample rate was chosen.

C. Selected hardware attacks

The selected hardware attacks are:

- **Meltdown**: among the most notorious hardware attacks, operates uniquely. While modern computers typically have the KPTI/KAISER mitigation against it, analyzing its behavior could be beneficial for the dataset. The Meltdown code was extracted from the IAIK GitHub repository [17].
- **Spectre V1, V2, and V4**: the infamous companion of Meltdown, has seen several versions released to date, with minor changes between them. A functional proof of concept (PoC) for Spectre V3 was not found, so it was skipped. Codes for the first [18], second [19], and fourth [20] versions of PoCs have been obtained from GitHub repositories.
- **ZombieLoad [21]**: similar to Meltdown, it captures sensitive data accessed by a user on a machine, and has been shown to work even on Meltdown-safe computers. The PoC by IAIK can be found on their GitHub repository [22].
- **Fallout [23]**: akin to Meltdown, leaks data from the CPU pipeline's store buffer and is classified under Microarchitectural Data Sampling (MDS) attacks along with RIDL [24]. The PoC code for the Fallout attack is sourced from Tristan Hornetz's GitHub repository [25].
- **Crosstalk [26]**: another MDS attack, aims to leak information between CPU cores, making it unique as it utilizes multiple cores, unlike other attacks. The source code for the proof of concept used is also obtained from Tristan Hornetz's GitHub repository [27].

These attacks bring the total number of malicious programs to 7. Other hardware attacks, like RIDL, Foreshadow [28], and ForeshadowNG [29], among others, were not selected because they rely on features not present in the laptop's architecture, such as Intel TSX [30].

D. Selection of benignware

The other half of the dataset will be generated using benign programs to contrast the behavior of the hardware attacks. To maintain balance, an equal number of benign programs (7) have been chosen:

- **Matrix multiplier**: A simple C program that multiplies large amounts of integer numbers to stress the computational sections of the CPU.
- **stress -c**: This is from the Debian `stress` tool, which stresses the CPU computing unit by repeatedly performing square roots of random numbers [31].
- **stress -m**: Also from the same tool, this option stresses the memory unit by repeatedly running malloc() and free() [31].
- **MiBench Bitcount**:A benchmark from the MiBench suite under the automotive category [32] available on Embecosm's Github repository [33]. It performs a bitcounting benchmark algorithm that stresses the CPU.
- **STREAM**: The STREAM benchmark [34], known for measuring memory bandwidth, will be used to stress the memory unit. The source code is available in Jeff Hammond's Github repository [35].
- **bzip2**: This is a high-quality lossless data compressor, chosen for both computational and memory workloads [36]. It will compress a fixed file, specifically the FreeBSD ISO image, to ensure replicability [37].
- **FFmpeg**: A multipurpose audio and video tool, used as a benchmark and example of a common mixed workload [38]. In this case, it will decode the "Big Buck Bunny" animation [39], commonly used for video testing [40], [41], [42].

The benchmarks selected are chosen to have similar execution profiles as the attacks listed. They either stress the memory unit, the CPU computational units or a mix. This ensures that the model can reliably distinguish malicious memory usage from memory-intensive workloads; and similarly for computational units or a mix.

V. Experiments and results

The first architecture used for testing is based on the Intel Core i5-8250U processor. The system runs Debian 11 (Bullseye) with the Linux kernel version 5.10.0. The processor operates at a maximum clock frequency of 3.4 GHz and features a 4-core / 8-thread configuration. This CPU belongs to Intel's Kaby Lake R (8th generation) family and is built using a 14nm process. It includes the following cache hierarchy:

- L1 Data Cache: 128 KiB
- L2 Cache: 1 MiB
- L3 Cache: 6 MiB

TABLE I
DATASETS, SCENARIOS AND SAMPLES

Dataset name	Representative scenario	%Samples/Type	Total samples
Balanced	Both representative benign applications and attacks are available for training. System administrator knows representative applications and attacks. The ML method has both information on what is benign and malign to make its prediction.	50% benign 50% malicious	28,000
Benign-only	Only representative benign applications are available for training. System administrator only knows the representative applications running on the system. Any other application will be deemed malign. The ML method turns into an anomaly detection setup.	100% benign	14,000
Malicious-only	Only representative malign applications are available for training. System administrator only knows the representative malign applications that can target the system. Any other application will be deemed benign. The ML method turns into an anomaly detection setup.	100% malicious	14,000

TABLE II
PARAMETERIZATION OF THE METHODS USED IN THE BALANCED DATASET

Method	Kernel	Parameters
Naive Bayes	Gaussian	n.a
	Multinomial	
Decision Tree	n.a	entropy, max_depth=10, min_samples_leaf=1, min_samples_split=5
		entropy, max_depth=None, min_samples_leaf=1, min_samples_split=2
Random Forest	n.a	Same optimal parameters of decision tree with 100 decision tree estimators
SVM	Lineal	C=1000
	Polinomial (second degree)	C=100
	RBF (Detection)	C=10, γ=10
	RBF (Classification)	C=100, γ=10

As mentioned in subsections IV-C and IV-D, we have a total of 14 programs: 7 benign and 7 malicious. From each of these programs, we obtain 2,000 samples, totaling 14,000 benign plus 14,000 malign samples. To conduct the experiments, we group these samples into three datasets: benign-only, malicious-only and balanced (both benign and malign samples). Table I shows for each dataset, its intended representative scenario and the proportion and type of samples. In all cases, 80% of the samples from each dataset were used for model training and 20% for testing.

A. Balanced dataset: Attack detection

For the balanced dataset, all the methods described in the previous section have been studied, except for the One-Class method (as it does not apply). For each case, hyperparameter tuning was performed using GridSearchCV [43], [44]. Table II shows the resulting hyperparameters for each method and variants or kernels studied in each case in the balanced dataset.

Table III shows the accuracy of each method studied. As shown in the table, Naive Bayes is the worst performing unless as it fails to detect benign samples correctly. We analyzed the case an it is caused by the non-independence between the counters used. The other 3 ML methods perform similarly (above 99% accuracy) being the SVM with RBF kernel the method that gives the best results for detection (i.e. benign/malign decision). Table IV shows the accuracy, recall, precision and F1-Score of this case (together with the best performing mechanisms of the next subsections).

TABLE III
PERFORMANCE OF MACHINE LEARNING MODELS ON THE BALANCED DATASET (INTEL CORE I5-8250U)

Method	Kernel	Metric
Naive Bayes	Gaussian	90% to detection and classification, but with the presence of false positives in detection, which impairs the for FFmpeg samples.
	Multinomial	53.69% to detection with problems to detect benign samples. By modifying the dataset to eliminate non independent HPCs, accuracy improves up to 90,89%.
Decision Tree	n.a	99.85% (Detection)
		99.79% (Classification)
Random Forest	n.a	99.94% (Detection)
		99.89% (Classification)
SVM	Lineal	99% detection and classification
	Polinomial (second degree)	99.8% detection and classification
	RBF	99.9% detection and classification

B. Balanced dataset: Classification

Beyond detecting if our system is under attack, we may want to know what kind of attack are we suffering to take specific remedial actions. As listed in Section IV-C, the attacks under study focus on different parts of the CPU and specific actions could be taken in each case.

We used the same ML methods as in the previous section to evaluate the effectiveness of detecting each different malign attack and benign application. Table III shows the accuracy results and, again, SVM with the RF Kernel is the most accurate. Figure 4 shows the confusion matrix for the classification using this method. Classification is nearly perfect (just 10 out of 24000 samples are misclassified and 5 being between Spectre V1 and Spectre V2).

C. Benign-only and malign-only

For the completely unbalanced datasets (benign and malicious), we used the One-Class SVM method with parameter values $\gamma = 1$ and $\nu = 0.01$ (obtained through GridSearchCV [43], [44]). Figures 2 and 3 show the confusion matrices of each dataset for detection. In the scenario where One-Class

SVM is used to detect benign samples, there are many false positives because the model encounters unseen samples during training and misclassifies them as malware. Similarly, in the case of using One-Class SVM to detect malicious samples, there are also numerous false positives due to the model's inability to accurately identify samples from the FFmpeg program. Overall, the F1-score is still over 90% in both cases, but not 99,9% as it is for the balanced dataset.

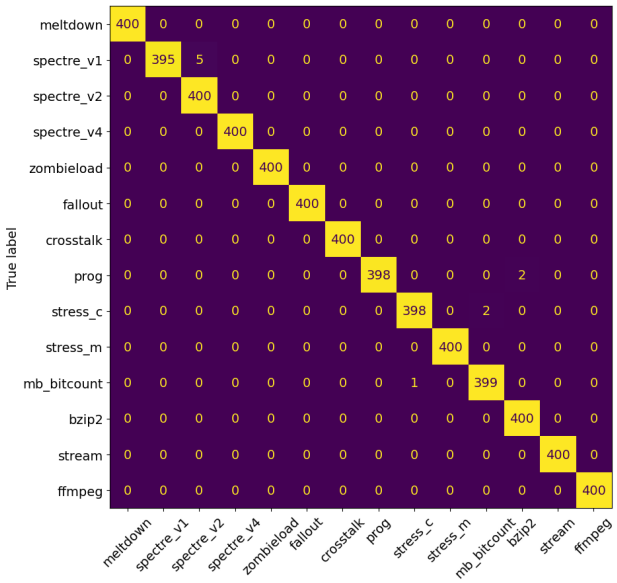


Fig. 1. Confusion matrix for classification using a RBF kernel SVM

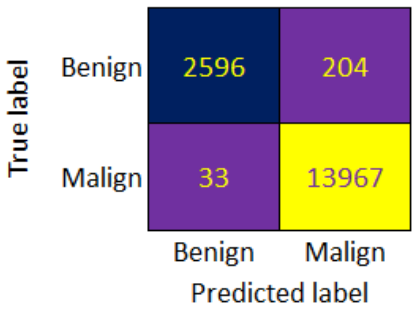


Fig. 2. Confusion matrix using a One-Class method and a benign dataset

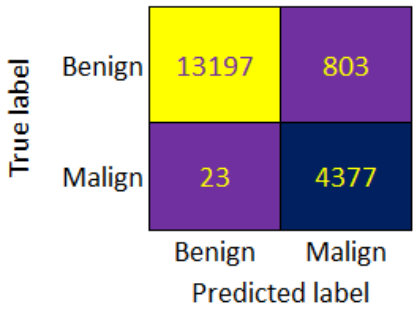


Fig. 3. Confusion matrix using a One-Class method and malicious dataset

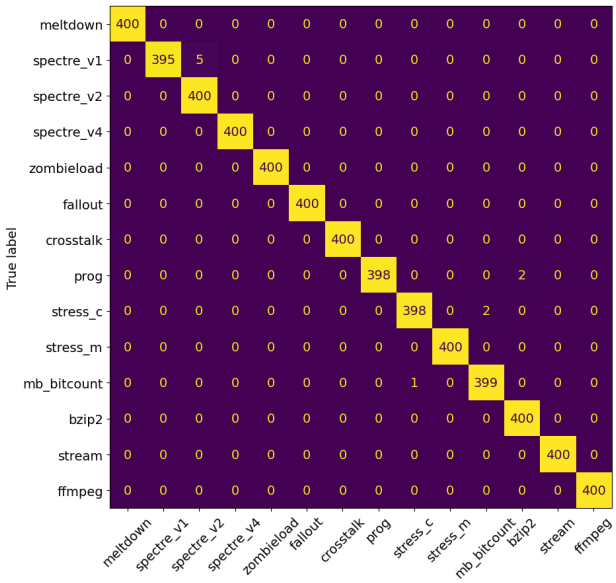


Fig. 4. Confusion matrix for classification using a RBF kernel SVM

TABLE IV
EVALUATION SUMMARY OF SVM-BASED MODELS ON INTEL CORE I5-8250U

Dataset	Method	Accuracy	Recall	Precision	F1
Balanced	SVM RBF (Detection)	99.96ª%	99.92%	100%	99.96%
	SVM RBF (Classification)	99.91%	99.91%	99%	99.91%
Benign	One-Class SVM	98.5%	92.71%	98.74%	95.63%
Malicious	One-Class SVM	95.51%	99.48%	84.5%	91.38%

D. Summary of results on Intel Core i5-8250U

Table IV displays accuracy, recall, precision and F1-score for both the SVM RBF method and the One-Class method. For the balanced dataset, all values are above 99,9%, indicating very precise predictions. In contrast, for the only-benign and only-malign datasets, the recall and precision values indicate the a higher presence of false positives. Thus, clearly performing behind the balanced dataset.

E. Cross-Architecture Experimental Validation

To assess the portability of our approach across different hardware platforms, we replicated the dataset generation process on a machine with a different architecture. The second architecture is based on the AMD Ryzen 7 3700X processor. This system also runs Debian 11 (Bullseye) but with a more recent Linux kernel version 6.1.0. The CPU has a base clock frequency of 3.6 GHz and can boost up to 4.4 GHz. It features 8 cores and 16 threads, offering significantly more parallel processing capabilities than the first architecture. The processor is part of AMD's Zen 2 architecture, manufactured using a 7nm process. Its cache configuration is as follows:

- L1 Data Cache: 256 KiB
- L2 Cache: 4 MiB
- L3 Cache: 32 MiB

The only required modification involved adapting one hardware performance counter (HPC): the original counter for last-level cache load misses, LLC-load-misses, was unavailable on the AMD processor. Instead, we substituted it with an equivalent counter, l3_comb_clstr_state.request_miss [45], which provides analogous information regarding cache miss behavior on this architecture.

Using this setup, we generated a new dataset equivalent in structure and size to the original. We then evaluated the previously trained models: RBF SVM and the malign-trained One-Class SVM (without retuning the hyperparameters), maintaining consistency with the original system. As in the baseline experiments, we applied the same post-processing steps for prediction smoothing and result refinement.

Table V presents the results obtained using the same methods previously applied to the Intel-based architecture (Table IV).The results confirm that the models exhibit similar behavior to that observed on the Intel platform.

TABLE V
SUMMARY OF SVM-BASED MODELS ON AMD RYZEN 7 3700X

Dataset	Method	Accuracy	Recall	Precision	F1
Balanced	SVM RBF (Detection)	99.96%	100%	99.92%	99.95%
	SVM RBF (Classification)	98.92%	98.72%	99.2%	98.95%
Benign	One-Class SVM	98.32%	92.6%	97.41%	94.94%
Malicious	One-Class SVM	94.41%	99.96%	75.1%	85.76%

The two architectures employed in this study exhibit substantial differences in computational capabilities and target design. The Intel Core i5-8250U is a low-power, 8th-generation mobile processor featuring 4 cores and 8 threads, optimized for energy-efficient operation in portable devices. Conversely, the AMD Ryzen 7 3700X is a high-performance desktop processor with 8 cores and 16 threads, manufactured using a more advanced 7nm process. It offers higher base and boost frequencies, as well as significantly larger cache capacities—most notably a 32 MiB L3 cache compared to 6 MiB in the Intel counterpart. These architectural distinctions position the Ryzen 7 3700X as more suitable for compute-intensive and parallelizable workloads, while the i5-8250U is better aligned with lightweight, general-purpose computing in mobile environments.

Despite these disparities, the experimental findings indicate that reproducing the complete workflow—including dataset generation and model evaluation—on an alternative hardware platform yields consistent and reliable results. The models under evaluation (RBF SVM and One-Class SVM) attained comparable levels of accuracy, even though they were initially trained and hyperparameter-tuned on the Intel-based system. Although a minor degradation in performance was observed, it was largely mitigated through post-processing techniques. This performance gap is attributed to the hardware-specific nature of the original hyperparameter optimization, which was tailored to the Intel architecture.

VI. CONCLUSIONS

This work builds a reliable and reproducible dataset by using hardware counters to generate samples through the execution of 14 programs in two x86 CPUs (Intel and AMD). It then evaluates various ML models to determine the most effective model for detecting and classifying hardware attacks. Among the models evaluated, the SVM with RBF kernel showed superior performance in detecting and classifying attacks. With an accuracy and F1-score over 99.9% for both detection and classification tasks.

We also analyzed two scenarios where only the benign applications are known and only the malign applications are known. In these scenarios, the One-Class ML model was used and was capable of achieving an F1-score above 90% in both cases. Yet, significantly below the 99,9% of the balanced dataset. The larger amount of false positives reduced the F1-score accordingly.

REFERENCES

[1] M. Lipp *et al.*, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium*, 2018.

[2] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy*, 2019.

[3] J. Corbet, *Kaiser: Hiding the kernel from user space*, https://lwn.net/Articles/738975/, Accessed: 13-05-2024.

[4] D. Gens, O. Arias, D. Sullivan, C. Liebchen, Y. Jin, and A.-R. Sadeghi, "Lazarus: Practical side-channel resilient kernel-space randomization," in *Research in Attacks, Intrusions, and Defenses*, Springer International Publishing, 2017, pp. 238–258.

[5] C. Canella, M. Schwarz, M. Haubenwallner, M. Schwarzl, and D. Gruss, "Kaslr: Break it, fix it, repeat," in *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 481–493. DOI: 10.1145/3320269.3384747.

[6] C. Li and J.-L. Gaudiot, "Detecting spectre attacks using hardware performance counters," *IEEE Transactions on Computers*, vol. 71, no. 6, pp. 1320–1331, 2022. DOI: 10.1109/TC.2021.3082471.

[7] S. Carnà, S. Ferracci, F. Quaglia, and A. Pellegrini, "Fight hardware with hardware: Systemwide detection and mitigation of side-channel attacks using performance counters," *Digital Threats*, vol. 4, no. 1, Mar. 2023. DOI: 10.1145/3519601.

[8] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016. DOI: https://doi.org/10.1016/j.asoc.2016.09.014.

[9] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing performance monitors for compromising keys of rsa on intel platforms," in *Cryptographic Hardware and Embedded Systems*, Springer Berlin Heidelberg, 2015, pp. 248–266.

[10] W. Kosasih, Y. Feng, C. Chuengsatiansup, Y. Yarom, and Z. Zhu, "Sok: Can we really detect cache side-channel attacks by monitoring performance counters?" In *19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 172–185. DOI: 10.1145/3634737.3637649.

[11] B. Otero Calviño, D. Andreu Gerique, and R. Canal Corretger, *Replication Data for: Hardware Attack detectoR via Performance counters analYsis Dataset (HARPY Dataset)*, version V1, 2025. DOI: 10.34810/data1982.

[12] C. D. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," 2008.

[13] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers - a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005. DOI: 10.1109/TSMCC.2004.843247.

[14] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[15] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, "Numerical Recipes 3rd edition: The Art of Scientific Computing," 2007.

[16] S. L. (speed47), *Spectre-meltdown-checker*, https://github.com/speed47/spectre-meltdown-checker, 2023.

[17] I. of Applied Information Processing and C. (IAIK), *Meltdown*, https://github.com/IAIK/meltdown.

[18] R. C. (crozone), *Spectrepoc*, https://github.com/crozone/SpectrePoC.

[19] A. C. (Anton-Cao), *Spectrev2-poc*, https://github.com/Anton-Cao/spectrev2-poc.

[20] Y. S. (mmxsrup), *Cve-2018-3639*, https://github.com/mmxsrup/CVE-2018-3639.

[21] M. Schwarz *et al.*, "ZombieLoad: Cross-privilege-boundary data sampling," in *CCS*, 2019.

[22] I. of Applied Information Processing and C. (IAIK), *Zombieload*, https://github.com/IAIK/ZombieLoad.

[23] C. Canella *et al.*, "Fallout: Leaking data on meltdown-resistant cpus," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019.

[24] S. van Schaik *et al.*, "RIDL: Rogue in-flight data load," in *S&P*, May 2019.

[25] T. H. (tristan-hornetz), *Fallout*, https://github.com/tristan-hornetz/fallout.

[26] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "CrossTalk: Speculative Data Leaks Across Cores Are Real," in *S&P*, Intel Bounty Reward, May 2021. [Online]. Available: https://download.vusec.net/papers/crosstalk_sp21.pdf.

[27] T. H. (tristan-hornetz), *Crosstalk*, https://github.com/tristan-hornetz/crosstalk.

[28] J. Van Bulck *et al.*, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proceedings of the 27th USENIX Security Symposium*, See also technical report Foreshadow-NG [29], Aug. 2018.

[29] O. Weisse *et al.*, "Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution," *Technical report*, 2018.

[30] *Intel® transactional synchronization extensions (intel® tsx) memory and performance monitoring update for intel® processors*, https://www.intel.com/content/www/us/en/support/articles/000059422/processors.html, Accessed: 28-05-2024, 2023.

[31] R. O. S. Projects, *Stress*, https://github.com/resurrecting-open-source-projects/stress.

[32] U. of Michigan, *Mibench version 1.0*, https://vhosts.eecs.umich.edu/mibench/, Accessed: 14-05-2024, 2002.

[33] Embecosm, *Mibench*, https://github.com/embecosm/mibench.

[34] J. D. McCalpin, *Stream: Sustainable memory bandwidth in high performance computers*, https://www.cs.virginia.edu/stream/, Accessed: 28-05-2024.

[35] J. H. (jeffhammond), *Stream*, https://github.com/jeffhammond/STREAM.

[36] *Bzip2*, https://sourceware.org/bzip2/, Accessed: 28-05-2024.

[37] *Parallel bzip2 compression benchmarks — openbenchmarking.org*, https://openbenchmarking.org/test/pts/compress-pbzip2, Accessed: 28-05-2024.

[38] *Ffmpeg*, https://ffmpeg.org/, Accessed: 28-05-2024.

[39] *Big buck bunny*, https://peach.blender.org/, Accessed: 28-05-2024.

[40] *Benchmark: Big buck bunny trailer*, https://dcpomatic.com/benchmarks/input.php?id=2, Accessed: 28-05-2024.

[41] *Ffmpeg rabbit benchmarks — openbenchmarking.org*, https://openbenchmarking.org/result/2311122-NE-FFMPEGRAB69, Accessed: 28-05-2024.

[42] *525.x264_r*, https://www.spec.org/cpu2017/Docs/benchmarks/525.x264_r.html, Accessed: 28-05-2024.

[43] P. M. Lerman, "Fitting segmented regression models by grid search," *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 29, no. 1, pp. 77–84, Dec. 2018. DOI: 10.2307/2346413. eprint: https://academic.oup.com/jrsssc/article-pdf/29/1/77/48620247/jrsssc_29_1_77.pdf. [Online]. Available: https://doi.org/10.2307/2346413.

[44] *Gridsearch documentation*, https://scikit-earn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV, Accessed: 13-05-2025.

[45] *Re: Amd zen2 l3$_m$issesevent*, https://www.spinics.net/lists/linux-perf-users/msg17608.html, Accessed: 14-07-2024.