Zero-Day Hardware-Supported Malware Detection of Stack Buffer Overflow Attacks: An Application Exploiting the CV32e40p RISC-V Core

Zero-Day Hardware-Supported Malware Detection of Stack Buffer Overflow attacks: an application exploiting the CV32e40p RISC-V core

Cristiano Pegoraro Chenet
Control and Computer Eng. Dep.
Politecnico di Torino
Torino, Italy
ORCID: 0000-0003-3974-9310

Alessandro Savino
Control and Computer Eng. Dep.
Politecnico di Torino
Torino, Italy
ORCID: 0000-0003-0529-7950

Stefano Di Carlo
Control and Computer Eng. Dep.
Politecnico di Torino
Torino, Italy
ORCID: 0000-0002-7512-5356

***Abstract*—The RISC-V architecture has become increasingly popular due to its open-source nature and flexibility, making it susceptible to various security attacks, such as Stack Buffer Overflow (SBO) attacks. It is crucial to effectively detect these attacks to ensure the security of RISC-V systems. This study explores Hardware-Supported Malware Detection (HMD) techniques for identifying these security threats. Using the CV32e40p RISC-V platform, we conducted simulations to assess the effectiveness of anomaly-based HMD methods in detecting SBO attacks by analyzing hardware microarchitectural events. The findings demonstrate the potential of these approaches but also reveal significant challenges in detection performance. These elements are essential for advancing the security capabilities of HMD systems in RISC-V environments.**

***Index Terms*—Cybersecurity, malware, hardware-based detection, RISC-V**

I. Introduction

Malware significantly threatens computer security. It includes any code modification within a software system aimed at causing harm or disrupting the system's intended function [1]. From personal computers, mobile phones, Internet of Things (IoT), 5G devices, to Cyber-Physical Systems (CPSs), computing systems are all vulnerable to malware. The complexity and size of modern systems, often indicated by a rising number of lines of code, amplify the threat. Factors such as numerous bugs, unsafe programming languages, improper configuration, and the ease of concealing malicious code create potential vulnerabilities. Additionally, the increased network connectivity expands the security risks, making all devices potential targets for attackers.

One of the most common exploitation methods malware employs is memory corruption [2]. Often, it leads to control-flow attacks [3] that are common, easy to construct, and demand minimal application-specific knowledge. They exploit vulnerabilities like Stack Buffer Overflows (SBOs) [4], or injection attacks to redirect the program's execution flow, enabling arbitrary code execution.

Literature extensively discusses various approaches to detecting malware [5]. In the early 2010s, researchers introduced the concept of Hardware-Supported Malware Detection (HMD) [6], [7]. This method entails dynamically analyzing microarchitecture events in a processor, i.e., Hardware Performance Counters (HPCs), using Machine Learning (ML) algorithms to distinguish between benign applications and malicious software. The transition to HMD is supported by its notable advantages: the potential for runtime detection, adaptability to code variations and unidentified malware, resistance to malware attempting to bypass protection mechanisms, and reduced detection costs [8]. A branch of HMD includes using anomaly detection techniques [9], where a classifier is trained on benign applications to identify attacks based on non-conforming data. Anomaly detection in HMD offers the advantage of not requiring a malware dataset for training, allowing for zero-day malware detection [10].

This paper aims to investigate the capability of HMD to support zero-day detection of SBO attacks. As a target platform, we use RISC-V since the platform is emerging, and the potentiality of its Performance Monitoring Unit (PMU) is still not fully explored in this domain [11], [12], [13]. Among the available RISC-V platforms, we selected the CV32e40p [14] since it is one of the most used, and it offers a simulator that includes the PMU.

The paper is organized as follows: Section II mainly reviews previous anomaly HMD works. Section III presents the methodology. Section IV discusses the experimental results, and section VI provides final considerations.

II. Background

SBO attacks employ memory corruption vulnerabilities, one of the most common malware exploitation methods. One common way of causing memory corruption is through buffer

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU and by the Vitamin-V project (Project number: 101093062) funded by the European Union. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union or the HaDEA. Neither the European Union nor the granting authority can be held responsible for them.

978-1-6654-7763-5/25/$31.00 ©2025 European Union

overflows. This happens when a program writes more data to a buffer than it can hold, which overwrites the adjacent memory space. The ultimate goal is redirecting the program's execution flow to malicious code. Specifically, the SBO attack is characterized by nonvalidated input overflowing a buffer allocated in the memory stack. Figure 1 shows the canonical SBO attack, where the function return address in the stack is overwritten, deviating the execution to a malicious function.

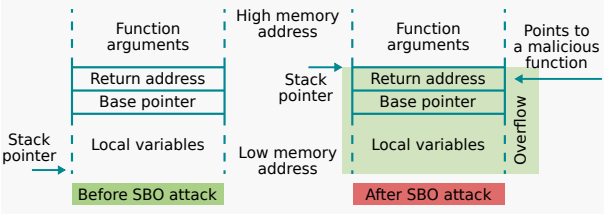

Fig. 1. Common SBO attack flow.

Several contributions investigated anomaly HMD, and the most relevant are summarized in Table I. While not considering any RISC-V architecture, they focus on the classifier. Moreover, they all lack a thorough investigation of different classifiers, nor do they perform a feature selection among the many HMD at their disposal. For this reason, the number of HPCs used as features varies a lot. Realistically, the reported wide variability in accuracy, can be linked to the lack of feature selection.

In detail, the detection implemented by [15] employed the One-class Support Vector Machines (OC-SVM) classifier. OC-SVM belongs to the family of Support Vector Machines (SVM) classifiers, a statistical approach that aims to find a decision boundary (hyperplane) that best separates data points belonging to different classes [16]. Observing a difficulty in distinguishing benign applications and malware from data due to the slight deviations produced, [15] applied Power Transform [17] before classification with SVM, making the data more Gaussian-like, thus enabling to detect security attacks.

TABLE I
RELEVANT STUDIES IN ANOMALY HMD, WITH BEST-CASE ACCURACY (A) AND AREA UNDER THE CURVE (AUC).

Year	Ref.	Target malware/exploit	#HPC	Class.	A	AUC
2014	[15]	ROP[1] on Internet Explorer	4	OC-SVM	-	1.00
		ROP on Adobe PDF Reader	4	OC-SVM	-	1.00
2015	[18]	ROP & SBO on web browser	6	LOF	-	-
2017	[19]	Rootkits	16	OC-SVM	0.75	-
2022	[20]	Data-only exploits	50	OC-SVM	0.91	-
		Data-only exploits	6	OC-SVM	0.75	-
		Data-only exploits	50	LZ78	0.92	-
		Data-only exploits	6	LZ78	0.84	-
2022	[21]	Stealthy attack on power grid	6	OC-SVM	0.94	-

[1] ROP is Return-Oriented Programming.

The Local Outlier Factor (LOF) classifier [22] is employed in [18] as an alternative to eliminate the necessity of Power Transform reported by [15], achieving good results. LOF tries to find anomalous data objects based on the concept of a local density deviation of a data point concerning its neighbors.

Remaining studies in Table I ([19], [20], [21]) also relied on OC-SVM, except [20], which used the LZ78 compression algorithm as a classifier, by collecting the HPCs, transforming them into symbols and then applying the algorithm to obtain a tree (which is the structure built for guiding the compression) and using it as pattern detectors.

III. METHODOLOGY

Assessing the effectiveness of HMD requires a specific pipeline, as shown in Figure 2. We employed a simulation environment (the red box) composed of a Controller script, which mounts the attack into the target applications, and a microarchitectural simulator, i.e., `GVSoC` [23], to run and collect the HPCs. The coupled analysis framework in green implements the entire pipeline from HPC extraction down to classification.

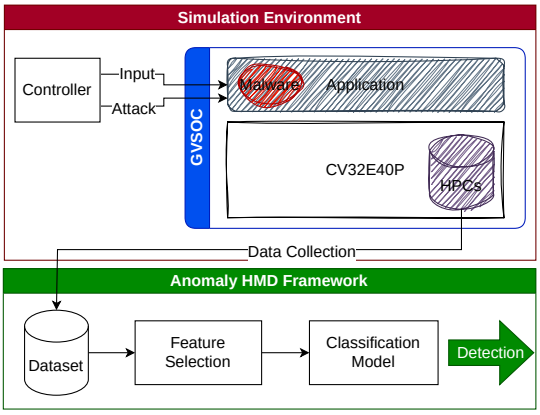

Fig. 2. Methodology overview: the simulation environment, in red, composed of the controller script and the `GVSoC` simulator, and the HMD framework, in green.

Instead of sampling the HPCs multiple times throughout the application's execution (like in [15], [18], [20], [21], from Table I), they are only collected at the end. ML algorithms are trained using the executions with random inputs, allowing for attack detection with any input. This method makes detecting subtle software behavior changes difficult but minimizes the number of ML inferences. Moreover, this method is viable for applications with a defined endpoint. In other cases, methods based on time or event domains are necessary [24].

A. Simulation environment

The control scripts manage the experiments by (i) mutating the application under attack to include the malware, (ii) injecting random input stimulus in the application, (iii) compiling it to run on the simulator, and (iv) running the simulation. The mutation is done by inserting the malware function and selecting where the SBO must be triggered.

To analyze different attack scenarios, three execution modes are implemented (see Figure 3): (i) Benign, where no malware is present; (ii) Overt, where, once the attack succeeds, the control passes to the malware, never returning to the original caller and (iii) Stealthy, in which a final jump back to the caller allows the malware to hide its presence.

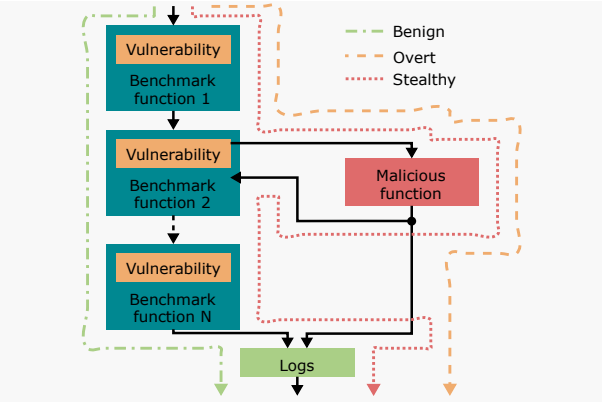

Fig. 3. Target application execution modes.

B. HMD framework

The HMD framework consists of feature extraction, feature selection, classification model training, and performance analysis.

1) Feature extraction: The feature extraction step aims to extract the HPCs and is performed through the Application Programming Interface (API) provided by the simulator. As the target is a zero-day detection, three different datasets are generated: (i) benign data used for training, (ii) stealthy executions, and (iii) overt executions. The last two are composed of benign data and malignant data. In each execution, the input stimulus is randomized.

2) Feature selection: Research has shown that certain HPCs are frequently overlooked or discarded due to their limited value in providing insights into program behavior [6], [25]. For instance, counters that record the total number of cycles or count active cycles do little more than track time without revealing meaningful information about the performance and efficiency of the analyzed code. Based on this prior knowledge from the literature, the first step in the feature selection was a manual pruning on the complete list of extracted HPCs, discarding `CYCLES` and `ACTIVE_CYCLES` counters, together inactive (zero-locked) HPCs. This pruning process results in a streamlined set that retains only those counters deemed most indicative of performance issues, strengthening detection of program anomalies and significantly reduceing the complexity of the analysis.

To further improve the quality of the detection, a feature selection process is employed, utilizing Principal Component Analysis (PCA) and a ranking method tailored to the specific needs of each target application. This ensures that the remaining HPCs provide the most valuable insights for performance evaluation. Additional details and the results of this analysis can be explored in Section IV, for the specific target applications considered in this study.

3) Classification models: This study analyzes OC-SVM, LOF, Isolation Forest, and Elliptic Envelope. While the first two classifiers are well known by previous work, Isolation Forest [26] and Elliptic Envelope [27] are two well-known anomaly detection algorithms. Still, they were never employed in HMD. Isolation Forest isolates anomalies as instances requiring fewer splits to isolate in the feature space. Elliptic Envelope creates an elliptical boundary around the data, considering anything outside the boundary as an anomaly.

IV. Results

The CV32e40p is a 32-bit RISC-V processor core with four pipeline stages that implement the RV32IM[F—Zfinx]C RISC-V specification and Parallel Ultra Low Power (PULP) custom extensions [14]. As mentioned, `GVSoC` [23] was used as a simulator. It is an event-driven simulator with a hardware-oriented description. The microprocessor supports tracking around ten HPCs, and the `GVSoC` simulator models all of them.

Target applications come from the MiBench suite [28] and are C applications compiled via the PULP Toolchain to run in bare-metal in `GVSoC`. The control scripts are written in Python and run in the host Operating System (OS). The target applications are annotated manually with (i) artificial vulnerabilities to enable the SBO exploitation, (ii) an additional code snippet to emulate a malicious code, and (iii) a function to log the HPCs. The list of applications under test comprises Advanced Encryption Standard (AES), Rivest–Shamir–Adleman (RSA) encryption, Secure Hash Algorithm (SHA), and the Dijkstra algorithm. They are all applications that do not require interaction with the I/O system, which is not very well emulated by `GVSoC`.

The vulnerabilities are functions with a C `memcpy` instruction followed by a buffer overflow and inserted arbitrarily before the call of some functions that compose the application. The Fibonacci number generator was employed to emulate a malicious code. It generates N Fibonacci numbers. It is important to note that detecting specific malware operations is not the primary goal. However, detection performance is influenced by the size of the malicious code. In fact, few extra instructions are introduced, detecting the SBO attack becomes more challenging due to minimal deviations in the HPCs. The Fibonacci function allows the production of different versions of the same code that vary in size by simply increasing or decreasing the sequence size. Thus, the control scripts can impose different ratios between the application and the malware, evaluated using the number of instructions executed (via `INSTR` HPC), by selecting the length of Fibonacci numbers generated. In the experimental results, two different sizes of the malicious code have been studied: 1% and 10% of the application. The dimension of the benchmarks is estimated as a function of the input (in number of bytes for the AES and SHA) and nodes (for the Dijkstra). It is worth noticing that, in the RSA benchmark, the estimation is more difficult because it depends on the random prime numbers generated by the algorithm. For this reason, a first execution is needed to measure the size, and then a second one allows data extraction.

For each application and attack execution mode (Stealthy and Overt), we ran twenty thousand executions (10K for training and 10K for testing), randomly varying the benchmark

input stimulus. In the testing, 50% of executions are in Benign mode, and 50% are with attacks.

Eventually, all experiments run on a host computer with Ubuntu 20.04.6 LTS (Focal Fossa) in different hardware configurations: AMD Ryzen 7 5700U and AMD Ryzen 9 7950X (for fast simulations).

As a performance evaluation metric, only accuracy is shown due to page constraints. Precision, recall, specificity, F1-score and AUC were also evaluated and did not reveal any novelty related to the accuracy, meaning that the metric shown is, in our case, properly representative of the performance.

A. Feature selection

Table II presents the HPCs ranked according to PCA. The ranking for a given target application is the same, independent of the execution mode (Overt or Stealthy). The `INSTR`, `RVC`, and `LD` HPCs are, respectively, the most significant HPCs for all the applications. Using this classification, the experimental results include, for each tested configuration of application and classifier, an incremental number of HPCs involved as input feature.

TABLE II
HPCs RANKED ACCORDING TO PCA. INDEX 1 DEFINES THE MOST SIGNIFICANT COUNTER.

HPC	Description	AES	RSA	RSA[1]	SHA	Dijkstra
INSTR	Number of instructions executed	1	1	1	1	1
LD_STALL	Number of load data hazards	4	7	6	4	5
LD	Number of data memory loads executed	3	3	3	3	3
ST	Number of data memory stores executed	5	5	5	5	6
JUMP	Number of unconditional jumps (j, jal, jr, jalr)	6	8	8	6	8
BRANCH	Number of branches (taken and not taken)	7	4	4	7	4
BTAKEN	Number of taken branches	8	6	7	8	7
RVC	Number of compressed instructions executed	2	2	2	2	2

[1] Refers to RSA implementation with constant prime numbers.

B. Detection performance in Stealthy mode

Figure 4 presents the detection accuracy for Stealthy mode. For each benchmark and classifier, four bars refer to accuracies based on 1, 2, 4, and 8 HPCs (from left to right in the figure, respectively). When using one, it employed the HPC with index 1 in Table II (in the case, `INSTR`); when using two, the HPCs with indexes 1 and 2 (in the case, `INSTR` and `RVC`), and so on. Analyzing the overall performance in AES, SHA, and Dijkstra, the accuracy is higher than 90% even when the size of the malicious code is 1% of the benchmark. Despite this good performance, attacks in RSA code are detected only when the ratio reaches 10%. Such performance is due to the algorithm's random search for prime numbers (for the public and private keys generation), which generates massive variability in the HPC even with the same inputs. To confirm the hypothesis, we performed the classification on a modified version of the algorithm where the prime numbers are constant. The results are in Figure 5, where the classification presents an accuracy close to 100%, also demonstrating that randomness in the application algorithm might impact HMD.

Figure 6 visually investigates the detection performances on the RSA using the PCA decomposition when malicious code size is 1%. The clear separation between normal and attack data in RSA with constant prime numbers, in contrast, the partial overlap in the original RSA demonstrates why performances improve. Despite not being plotted, similar results can be appreciated for the other applications.

After carefully analyzing the four classifiers, it is evident that the LOF classifier is a powerful tool. Unlike the others, it consistently achieves an accuracy close to or higher than 90% when dealing with a small number of HPCs. For instance, it performs exceptionally well with AES, RSA (using constant prime numbers), and SHA with just one HPC and a size of 1%. Additionally, the Elliptic Envelope also achieves accuracies higher than 90% in more challenging cases that require detection of multiple HPCs. On the other hand, the Isolation Forest performed poorly and could not detect attacks. This poor performance can be attributed to the masking problem, where numerous anomalies hide their presence, making it difficult for the classifier to isolate each anomaly effectively [26], [29].

C. Detection performance in Overt mode

In this execution mode, the results are also expressed as a function of the size of the malicious code. As the control scripts randomly choose where to launch the attack, and as in the Overt mode, the program finishes after the execution of the attack, the application may not complete the processing of its input data. Consequently, the ratio between the size of the malicious code and the benchmark may, eventually, end up bigger than the nominal stated value.

The detection accuracy for the Overt mode is presented in Figures 7 and 8. In AES, RSA, SHA, and Dijkstra, the accuracy is at least 94% when the malicious code size is 1%. The standard RSA implementation only achieved 65% accuracy. With a malicious code size of 10%, the accuracy improves significantly, reaching 99% with eight HPCs.

Compared to the Stealthy mode, the performance is slightly better in Overt mode. For instance, in Overt mode, accuracy is notably higher in cases such as AES (OC-SVM and Isolation Forest) reaching close to 70% accuracy, SHA (Isolation Forest) reaching close to 80% accuracy, and SHA (OC-SVM) and Dijkstra (LOF) reaching close to 90% accuracy. However, the performance shows a reduction if compared to Stealthy mode in a few cases, such as in the AES application (LOF classifier). The overall better performance in Overt mode is attributed to the possible cases in which the benchmark does not complete its execution, significantly impacting the HPCs and improving the detection.

When comparing classification algorithms, LOF and Elliptic Envelope performed the best again. Also, in this case, the RSA algorithm results in a more critical situation due to the same prime number randomization observed in Stealthy mode. Figure 9 shows the PCA decomposition in Overt mode, which confirms a similar overlapping of the three principal components that influence the detection.

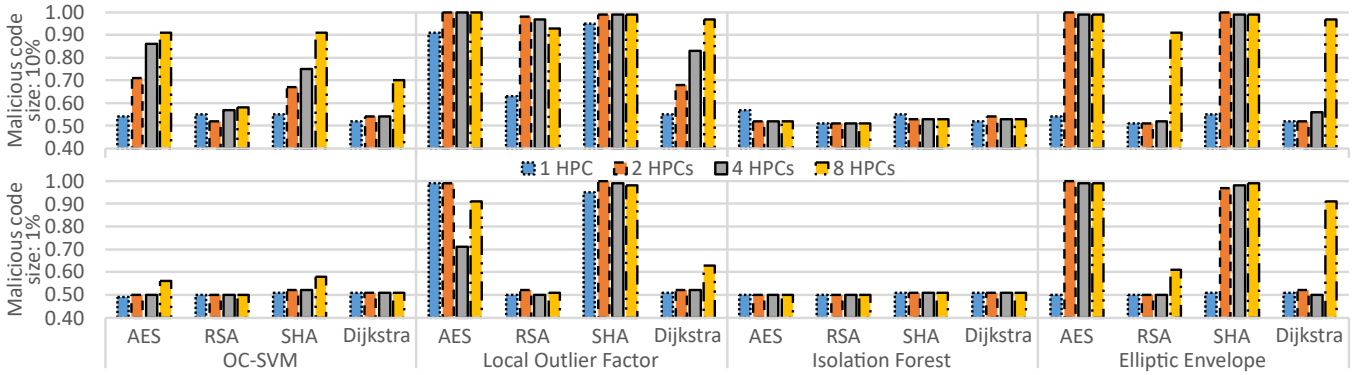

Fig. 4. Accuracy for Stealthy mode.

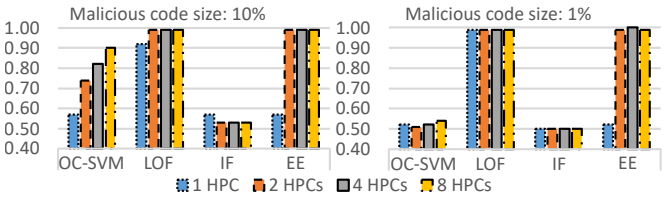

Fig. 5. Accuracy for RSA with constant prime number (Stealthy mode). IF is an Isolation Forest, and EE is an Elliptic Envelope.

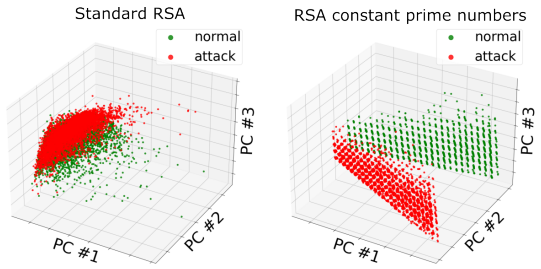

Fig. 6. RSA related PCA decomposition in the three first principal components for the Stealthy mode.

Eventually, we evaluated the capability of training a single detection model by training all classifiers with all target applications in the Overt mode. However, the final detection performance was poor. The best accuracy is 83% using a LOF to classify a 10% malicious code size and 78% for a 1% size malicious code. This suggests that using specialized detectors for each application is the most efficient solution.

Overall, obtained results confirm that a zero-day detection of SBOs attacks is feasible, but probably an enhanced PMU tracking specific events might be necessary.

V. Code and Data Availability

The code and data supporting this study and required to reproduce all published results are publicly available on GitHub at https://github.com/smilies-polito/ZeroDay_HWBasedMalwareDetection_SBOAttacks/releases/tag/v1.0.

VI. Conclusions

This work examined the effectiveness of anomaly HMD approaches in detecting security attacks, particularly SBO attacks. These approaches have potential for identifying other attacks, including zero-day malware. The findings indicate that analyzing eight HPCs enhances detection performance compared to fewer HPCs. RISC-V designers can improve real-time detection by supporting multiple counters. Additionally, HMD is more effective when detectors are tailored to specific applications, highlighting a trade-off between performance and protection costs. However, challenges remain, such as the unpredictability of protected applications, as seen with RSA, which offers opportunities for future research.

Overall, HMD approaches can significantly benefit RISC-V architectures, especially if the next-generation PMU incorporates specific event tracking. A promising solution involves using software and hardware detectors, with hardware as the primary defense.

References

[1] G. McGraw and G. Morrisett, "Attacking malicious code: A report to the infosec research council," *IEEE Software*, vol. 17, no. 5, pp. 33–41, 2000.

[2] V. van der Veen, N. dutt Sharma, L. Cavallaro, and H. Bos, "Memory errors: The past, the present, and the future," in *Research in Attacks, Intrusions, and Defenses*, D. Balzarotti, S. J. Stolfo, and M. Cova, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 86–106.

[3] W. P. Arthur, "Control-flow security," Ph.D. dissertation, University of Michigan, Michigan US, 2016.

[4] A. One, "Smashing the stack for fun and profit," *Phrack*, vol. 7, no. 49, 1996.

[5] T. Alsmadi and N. Alqudah, "A survey on malware detection techniques," in *2021 International Conference on Information Technology (ICIT)*, 2021, pp. 371–376.

[6] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing*, ser. STC '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 71–76. [Online]. Available: https://doi.org/10.1145/2046582.2046596

[7] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, jun 2013. [Online]. Available: https://doi.org/10.1145/2508148.2485970

[8] C. P. Chenet, A. Savino, and S. Di Carlo, "A survey on hardware-based malware detection approaches," *IEEE Access*, vol. 12, pp. 54 115–54 128, 2024.

[9] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, jul 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

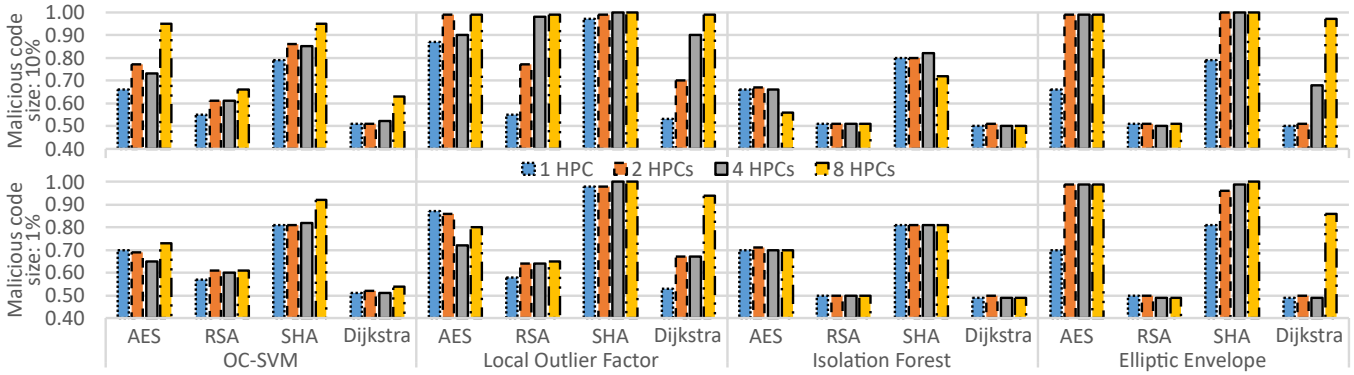

Fig. 7. Accuracy for Overt mode.

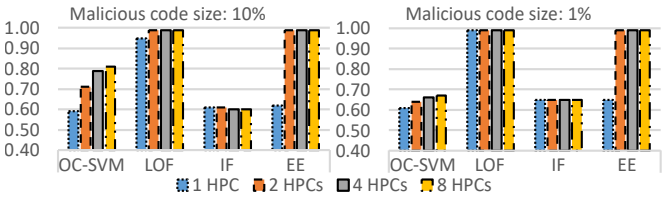

Fig. 8. Accuracy for RSA with constant prime number (Overt mode). IF is an Isolation Forest, and EE is an Elliptic Envelope.

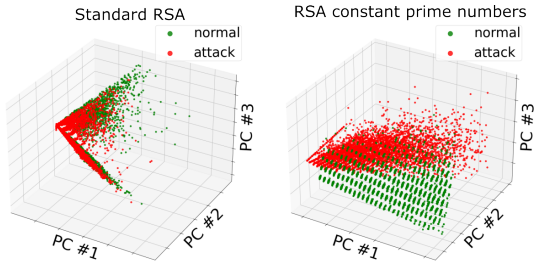

Fig. 9. RSA related PCA decomposition in the three first principal components for the Overt mode.

[10] Z. He, T. Miari, H. M. Makrani, M. Aliasgari, H. Homayoun, and H. Sayadi, "When machine learning meets hardware cybersecurity: Delving into accurate zero-day malware detection," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 85–90.

[11] R. Canal *et al.*, "Vitamin-v: Virtual environment and tool-boxing for trustworthy development of risc-v based cloud services," in *2023 26th Euromicro Conference on Digital System Design (DSD)*, 2023, pp. 302–308.

[12] M. Alonso *et al.*, "Validation, verification, and testing (vvt) of future risc-v powered cloud infrastructures: the vitamin-v horizon europe project perspective," in *2023 IEEE European Test Symposium (ETS)*, 2023, pp. 1–6.

[13] ——, "Special session: Security and ras in the computing continuum," in *2024 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2024, pp. 1–6.

[14] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, A. Pullini, D. Rossi, E. Flamand, F. K. Gürkaynak, and L. Benini, "Near-threshold risc-v core with dsp extensions for scalable iot endpoint devices," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2700–2713, 2017.

[15] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Cham: Springer International Publishing, 2014, pp. 109–129.

[16] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[17] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, 2000. [Online]. Available: http://www.jstor.org/stable/2673623

[18] A. Garcia-Serrano, "Anomaly detection for malware identification using hardware performance counters," 2015.

[19] B. Singh, D. Evtyushkin, J. Elwell, R. Riley, and I. Cervesato, "On the detection of kernel-level rootkits using hardware performance counters," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 483–493. [Online]. Available: https://doi.org/10.1145/3052973.3052999

[20] G. Torres and C. Liu, "Where's waldo? identifying anomalous behavior of data-only attacks using hardware features," in *Proceedings of the 19th ACM International Conference on Computing Frontiers*, ser. CF '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 75–84. [Online]. Available: https://doi.org/10.1145/3528416.3530226

[21] C. Konstantinou, X. Wang, P. Krishnamurthy, F. Khorrami, M. Maniatakos, and R. Karri, "Hpc-based malware detectors actually work: Transition to practice after a decade of research," *IEEE Design & Test*, vol. 39, no. 4, pp. 23–32, 2022.

[22] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: Association for Computing Machinery, 2000, p. 93–104. [Online]. Available: https://doi.org/10.1145/342009.335388

[23] N. Bruschi, G. Haugou, G. Tagliavini, F. Conti, L. Benini, and D. Rossi, "Gvsoc: A highly configurable, fast and accurate full-platform simulator for risc-v based iot processors," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, 2021, pp. 409–416.

[24] B. Sprunt, "The basics of performance-monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64–71, 2002.

[25] N. Patel, A. Sasan, and H. Homayoun, "Analyzing hardware based malware detectors," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[26] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[27] P. J. Rousseeuw and K. V. Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1999.10485670

[28] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*, ser. WWC '01. USA: IEEE Computer Society, 2001, p. 3–14.

[29] T. Liu, Z. Zhou, and L. Yang, "Layered isolation forest: A multi-level subspace algorithm for improving isolation forest," *Neurocomputing*, vol. 581, p. 127525, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231224002960