Malware Detection Using Machine Learning Algorithms Based on Hardware Performance Counters: Analysis and Simulation

Omar Bawazeer¹, Tarek Helmy¹, *, and Suheer Al-hadhrami²

¹Department of Information and Computer Science, College of Computer Sciences and Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.
²Department of Computer Engineering, College of Engineering and Petroleum, Hadhramout University, Al-Mukalla, Yemen.
Emails: g201407380@kfupm.edu.sa, helmy@kfupm.edu.sa, and S.alhadhrami1@gmail.com

Abstract. In the last decade, Hardware Performance Counters (HPCs) events are increasingly used by Machine Learning (ML) algorithms for malware detection. Modern processors provide a variety of HPCs to measure and monitor processes' events such as memory accesses, instructions, etc. during their execution. In this paper, an analysis study to categorize the machine learning algorithms based on HPCs that have been used for malware detection is introduced. Besides, the most efficient and effective features of HPCs that have been exploited to recognize the abnormal activities on various systems are identified. Furthermore, the Neural Network (NN) algorithms including Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Full Order Radial Basis Function (RBF) algorithms are used to simulate several experiments from the literature. The simulation results show that the accuracy of MLP, CNN, and Full Order RBF are 96.95%, 98.22%, and 98.68%, respectively.

1. Introduction
Malicious software or malware is designed to harm or damage the system or some of its resources. There are various types or families of malware such as worms, viruses, Trojan, ransomware, spyware, adware, etc. Recently, there are many sophisticated techniques used to hide malicious activities in the victims' devices from detection by antiviruses and protection systems. This leads the researchers to look for new techniques, methods, approaches, and tools to improve the detection and protection systems against malware. Generally, detection techniques of malicious software can be categorized into three main types which are Signature-based, Anomaly-based, and Heuristic-based techniques. Signature-based techniques are basically used by most antivirus applications. Each antivirus has a database of patterns for well-known malware and if any application matches any one of these patterns, it will consider as malware. Whereas the Anomaly-based techniques depend on monitoring the system behavior to detect abnormal activities. In addition, the most recent technique is Heuristic-based. The researchers use Heuristic-based techniques to take the advantage of the previous types through implementing new sophisticated algorithms and mechanisms such as machine learning and neural networks to detect and identify malicious and abnormal activities in the system [18-20]. In the last decades, Intel and AMD other modern processors provide a variety of Hardware Performance Counters (HPCs) to measure and

*On leave from College of Engineering, Computers & Automatic Control Engineering Department, Tanta University, Egypt
monitor events through the execution of processes related to memory accesses, instructions, etc. Recently, many researchers have started to use HPCs to monitor several systems activities in order to differentiate between normal and abnormal behavior in that system. The supported events by hardware performance counters can be categorized into two types: architectural events or non-architectural (micro-architectural) events. The common hardware events of performance counters on various types of processor architectures such as cycles and instructions known as architectural events. Whereas the specific events of certain processor architectures are known as non-architectural or (micro-architectural) events [10].

HPCs events and features are increasingly used for malware detection and other security purposes. To the best of our knowledge, from the literature, there is no recent survey on malware detection using machine learning based on HPCs events. So, for all of that, we conduct a survey to discover the malware detection techniques that are currently used and the most efficient and effective features that have been exploited to enhance the security of the computing systems. As well as to categorize these techniques. The contributions of this paper include categorizing the malware detection using machine learning algorithms based on HPCs events to disclose malicious activities for various systems, processes, or special types of attacks. Also, we identify the most efficient events of HPCs correlated to security issues by tracing and documenting the selected features in the literature that are used for malware detection techniques. Noteworthy, we recognize the most recent algorithms as well used for malware detection based on HPCs. Finally, we have conducted several experiments using MLP, CNN, and Reduced Order RBF to simulate the work that has been done on [1].

The rest of this paper is organized as follows: Section 2 presents the literature review including categorizing malware detection using machine learning algorithms based on HPCs, HPCs features used for malware detection, and machine learning techniques used for malware detection Based on HPCs. In Section 3 and 4, we introduce the experiment setup and dataset. Then, the results and discussion are presented in Section 5. Finally, we concluded the paper and set the future work in Section 6.

2. Literature Review

2.1. Categorizing Malware Detection Using Machine Learning Algorithms Based on HPCs

![Figure 1 Categorization of Existing Malware Detection Using HPCs Features](image)

We can categorize the existing malware detection research that uses machine learning algorithms based on HPCs features to identify malicious activities into four main classes. The first category uses the HPCs events to classify and differentiate between the malware and benign application. The second category uses the features of HPCs to monitor the anomalous behavior of systems or processes. The third category is malware detection in the embedded systems. In the last category, some researchers use HPCs features to detect special types of attacks. Figure 1 illustrates the categorization of existing malware detection using machine learning algorithms based on HPCs features.

- **Classification**: The main implementation of using HPCs features for detecting malware through machine learning algorithms is for classifying and identifying malware and benign applications. Most literature concentrated on what are the appropriate HPCs events (see Table 1) that can be used for malware detection and how many of them should they used. Another important point is related to the type of machine learning algorithm that should be used. For example, the authors of [2], [6], [8], [10], and [14] used 4, 7, 8, 6, and 5 machine learning algorithms, respectively,
to evaluate their results with the HPCs features. Other researchers used only one or two types of machine learning algorithms like [1], [3], [11], and [12]. Noteworthy, CNN achieved the best results amongst others (see figure 4).

- Anomaly Detection: Another implementation of HPCs events is to use the HPCs for detecting the anomalous activities in a system. For instance, the authors of [11] proposed an approach to use SVM algorithm with instruction and branch instructions of HPCs events for monitoring software in real-time of a cyber-physical system. Also, in [15], the authors introduced their approach for identifying anomalous activities based on measuring the HPCs events during program execution. They showed the ability of HPCs events to recognize the anomalous activities that can cause a system failure.

- Embedded Systems: Other researchers exploited the existence of HPCs in the embedded processors to use them for malware detection. In [4], a lightweight approach using hardware-assisted malware detection by machine learning is introduced in embedded devices. They applied their approach with several algorithms and they got 93.2% as the best accuracy with the SMO algorithm. Authors in [5] and [13] proposed a ConFirm to identify anomalously changes in the embedded control systems firmware by monitoring the low-level HPCs events that can be happened during the run time of firmware.

- Detect Special Types of Attacks: Furthermore, several papers introduced some approaches to use HPCs events to detect special types of attacks. The authors of [7] proposed an approach for identifying cross-VM cache-based side-channel attacks by utilizing fine-grained data of hardware and HPCs following the anomaly detection method of Gaussian. Wang and Karri in [9] introduced NumChecker, a monitoring virtual machine for identifying and detecting control-flow anomaly changing in kernel system call of rootkits in another virtual machine. This technique depended on counting a certain number of events of HPCs during the execution of system calls. Also, in [12], the authors used the HPCs with multiple machine learning classifiers to present Moving Target Defense (MTD) that used to detect the adversarial attacks.

![Figure 2 Distribution of HPCs Features Among Several Recently Published Research used for Malware Detection](image)

**Figure 2** Distribution of HPCs Features Among Several Recently Published Research used for Malware Detection

### 2.2. HPCs Features Used for Malware Detection

The most effective and crucial points for using machine learning algorithms is the features selection. So, to identify the most important features and select the appropriate features that can recognize the signatures and behavior of malware, we propose Table 1 based on our literature review. These features can be used by machine learning techniques to train the algorithms in order to detect malware. Table 1
shows the statistics of the HPCs features on most recent research in the literature during the last five years. Table 1 illustrates that the most features that are used in the published work are branch instructions, instructions, branch_misses, and cache references. The branch instructions are used on 10 research out of 13 whereas the instructions are utilized on 5 out of 13. In total, we counted 35 features used around 75 times in all references in this study. On average, each research uses around 5 features for detecting the malware. Figure 2 and Table 1 show the distribution of HPCs features among the recent published research used for malware detection.

Table 1. Distribution of HPCs Features Among Several Recently Published Research used for Malware Detection.

| Features                     | [1] | [2] | [3] | [4] | [5] | [6] | [7] | [8] | [10] | [11] | [12] | [13] | [14] | Σ   |
|------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|-----|
| branch instructions          | 1   | 1   | 1   | 1   | 1   | 0   | 1   | 0   | 1    | 0    | 1    | 1    | 1    | 10  |
| instructions                 | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 1    | 1    | 1    | 1    | 5   |
| branch_misses                | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 1   | 0    | 0    | 0    | 0    | 1    | 4   |
| cache references             | 1   | 1   | 0   | 1   | 0   | 0   | 0   | 0   | 0    | 1    | 0    | 0    | 0    | 4   |
| branch_loads                 | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 1    | 3   |
| cache misses                 | 1   | 1   | 0   | 0   | 0   | 0   | 1   | 0   | 0    | 0    | 0    | 0    | 0    | 3   |
| iTLB_load_misses             | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 1    | 3   |
| LLC_dcache_stores            | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 1    | 3   |
| LLC-load-mis                 | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 0   | 0    | 0    | 0    | 1    | 0    | 3   |
| LLC_store_misses             | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 1    | 0    | 0    | 3   |
| LLC_loads                    | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 1    | 0    | 0    | 2   |
| return instructions          | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 2   |
| store instructions           | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 2   |
| branch_load_misses           | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| bus-cycles                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 1    | 1   |
| dTLB_load_misses             | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| dTLB_store_misses            | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1    | 0    | 0    | 0    | 0    | 1   |
| dTLB_loads                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 1    | 0    | 0    | 0    | 1   |
| hardware instructions        | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 1   |
| Indirect                     |     |     |     |     |     |     |     |     |      |      |      |      |      |     |
| instructions                 |     |     |     |     |     |     |     |     |      |      |      |      |      |     |
| iTLB_loads                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 0    | 0    | 0    | 1   |
| iTLB-cache-misses            | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| iTLB-r-accesses              | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| LLC_dcache_load-misses       | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| LLC_stores                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 0    | 1    | 0    | 0    | 1   |
| LLC-misses                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| LLC_references               | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| mispredicted return          | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0    | 1    | 0    | 0    | 0    | 1   |
| instructions                 |     |     |     |     |     |     |     |     |      |      |      |      |      |     |
| node_loads                   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0    | 0    | 0    | 0    | 0    | 1   |
| store micro-operations       | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 1    | 0    | 0    | 0    | 0    | 1   |
| Total features per research  | 5   | 12  | 1   | 3   | 4   | 4   | 4   | 16  | 5    | 2    | 7    | 4    | 8    | 75  |
2.3. Machine Learning Techniques Used for Malware Detection Based on HPCs

We have also inspected the most recent techniques that are used for malware detection based on HPCs events. We found that machine learning algorithms are mainly used for that purpose. In this paper, we have traced the machine learning techniques in many types of research to recognize them and identify what kinds of algorithms are used. We summarize the distribution of machine learning techniques used for malware detection based on HPCs events in Table 2 and Figure 3. We found that the J48, JRip, MLP, and OneR algorithms are used more than others for malware detection.

| No. | Techniques       | [1] | [2] | [3] | [4] | [6] | [8] | [10] | [11] | [12] | [14] | Total Usage |
|-----|------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-------------|
| 1   | J48              | 0   | 1   | 0   | 1   | 0   | 1   | 1   | 0   | 0   | 1   | 5           |
| 2   | JRip             | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 0   | 0   | 1   | 4           |
| 3   | MLP              | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 0   | 1   | 4           |
| 4   | OneR             | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 0   | 0   | 1   | 4           |
| 5   | Decision Tree    | 0   | 0   | 0   | 0   | 1   | 0   | 1   | 0   | 1   | 0   | 3           |
| 6   | SMO              | 0   | 0   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 0   | 3           |
| 7   | SVM              | 0   | 0   | 0   | 0   | 1   | 0   | 1   | 1   | 0   | 0   | 3           |
| 8   | BayesNet         | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 2           |
| 9   | CNN              | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 2           |
| 10  | KNN              | 0   | 0   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 2           |
| 11  | Logistic Regression | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1   | 2           |
| 12  | Neural Network   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 1   | 0   | 2           |
| 13  | REPTree          | 0   | 0   | 0   | 1   | 0   | 1   | 0   | 0   | 0   | 2           |
| 14  | Naive Bayes      | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1           |
| 15  | Random Forest    | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 0   | 1           |
| 16  | SGD              | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 0   | 1           |
| 17  | Ibk              | 0   | 0   | 0   | 0   | 0   | 0   | 1   | 0   | 0   | 1           |
|     | Total Usage per Research | 1   | 4   | 1   | 7   | 7   | 8   | 6   | 1   | 2   | 5   | 42          |

However, the more important information here is not related only to the total number of algorithms' usage, it is actually related also to the accuracy of malware detection achieved by these algorithms. So, we have continued our work to investigate the effectiveness of the algorithms by recording their accuracy rates. We found that the most effective one is the CNN which achieved an accuracy of 97% on [1] and 95.5% on average. Next, the BayesNet, MLP, and J48 algorithms came after CNN and achieved accuracy on average 92.325%, 88.892%, 87.342%, respectively.

These results inspired us to use NN algorithms such CNN, MLP, and Full Order RBF to simulate the work of [1] which achieved the best results on the literature. Figure 4 summarize the distribution of accuracy results for machine learning algorithms used for malware detection based on HPCs.
3. Experiment Setup

In this Section, we discuss the experimental setup to conduct our simulation using three NN-based algorithms which are MLP, CNN, and Full Order RBF. The goal of executing these experiments is to simulate the work of Pattee and Lee in [1] by using a dataset having the same features (28*28*1) as in [1]. Moreover, to investigate the effectiveness of the RBF algorithm since it is not used before in the literature. To increase the efficiency and reduce the time consuming, we have executed the experiments of the CNN algorithm on the Google Colab [16] and the other algorithms on the HPC of KFUPM at ICTC [17].

4. Data Set

We have contacted several authors to get the dataset of HPCs. Unfortunately, we did not get any response from them. So, to imitate and simulate the work of [1], we have to select a dataset with the same specification features. For that, the dataset must be a dataset of grayscale images with a size of 28
The training dataset contains 60000 grayscale images whereas the testing dataset contains 10000 grayscale images. Noteworthy, the size of all images in both datasets is $28 \times 28 \times 1$ pixels. This means that every image has 784 ($28 \times 28 \times 1$) values with the actual value of the image in the last column of the dataset (784+1).

Each image in training and testing datasets contained one hand-written Arabic digits from 0 to 9. To get an accurate output value of each image, we have transformed each integer value (0-9) of the image into a binary number of ten digits. We select the output layer of 10 neurons to make the NN more flexible. We build a multi-class classifier consisting of 10 classes to classify digits 0 through 9. The output neurons are arranged in such a way that outputs from each one of them corresponds to the value of the digit 0 through 9. Table 3 shows the mapped values that we used in this project.

### Table 3. Mapping the Output Values

| Actual Output | Mapped Output |
|---------------|---------------|
| 0 1 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 1 0 1 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 2 0 0 1 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 3 0 0 0 1 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 4 0 0 0 0 1 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 5 0 0 0 0 0 1 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 6 0 0 0 0 0 0 1 0 0 | 0 0 0 0 0 0 0 0 0 0 |
| 7 0 0 0 0 0 0 0 1 0 | 0 0 0 0 0 0 0 0 0 0 |
| 8 0 0 0 0 0 0 0 0 1 | 0 0 0 0 0 0 0 0 0 0 |
| 9 0 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 |

As a result of the output layer of the NN, we select the maximum output as one digit and the others are mapped to zeros. Table 4 demonstrates an example to show how we map the actual outputs.

### Table 4. Mapping the Actual Outputs into Closest Integer

| Actual output | Mapped Output |
|---------------|---------------|
| 0.230         | 0             |
| 0.102         | 0             |
| 0.157         | 0             |
| 0.0031        | 0             |
| 0.0854        | 0             |
| 0.0364        | 0             |
| 0.991         | 1             |
| 0.0004        | 0             |
| 0.00097       | 0             |
| 0.561         | 0             |

As a result of the output layer of the NN, we select the maximum output as one digit and the others are mapped to zeros. Table 4 demonstrates an example to show how we map the actual outputs.

5. Results and Discussion

5.1. Multi-Layer Perceptrons Neural Network

The structure, training performance, and all the findings of the MLP NN algorithm are shown and summarized in Table 5. The MLP topology contains one hidden layer including 110 neurons and the
The tangent sigmoid activation function has been used. We have achieved around 98.81% and 96.95% as accuracy for training and testing phases, respectively. This experiment consumed around 2.785 minutes.

5.2. Convolutional Neural Network

In CNN, we have used the Relu activation function for the hidden layer with 128 neurons and the softmax activation function for the output layer. Table 5 summarizes the results of CNN. Here, we have got better results than MLP. CNN achieved 99.69% and 98.22% as accuracy for training and testing phases, respectively in around 4.85 minutes.

5.3. Full Order Radial Basis Function

Here, we have used the RADBAS Radial Basis activation function. We have used only 50000 out of 60000 as inputs to the Full Order RBF NN to avoid the “out of memory” error that happened when we used 60000 as inputs. Table 5 summarizes the results of the training and testing phases of Full Order RBF. We have achieved slightly better results than CNN. We got 100% and 98.68% as accuracy for training and testing phases, respectively. This experiment takes around 4.8525 hours.

Table 5. A Comprehensive Comparison of MLP, CNN, and Full Order RBF

| Algorithms & Dataset | Algorithms | MLP | CNN | FO_RBF |
|----------------------|------------|-----|-----|--------|
| No. of Inputs        |            | 28*28= 784 |     | -      |
| No. of Outputs       |            | 10  |     | -      |
| Execution Platform & Scripts type | Python | - | Google Colab | - |
|                      | MATLAB    | HPC |     | HPC    |
|                      | Python    | - | Tensorflow (keras) | - |
| ToolBox & Libraries  | MATLAB    | NN Toolbox (patternne) | - | NN Toolbox (newrb) |
| No. of Hidden Layers & (Neurons) | 1 (110) | 1 (128) | 1 (50000) |
| NN Topology | Activation Function | Tangent Sigmoid | Relu & Softmax | RADBAS Radial Basis |
| MSE for Training | 0.043  | 0.0097 | 0 |
| MSE for Testing | 0.1099 | - | 0.0026 |
| No. of pattern | Training | 60000 | 50000 |
|                | Testing  | 10000 | 10000 |
| Correct Estimation | Training | 59285 | 59814 | 50000 |
|                   | Testing  | 9695 | 9822 | 9868 |
| Performance | Training | 98.81% | 99.69% | 100% |
|               | Testing  | 96.95% | 98.22% | 98.68% |
| Execution time | ≈ 2.785 m | ≈ 4.85 m | ≈ 4.8525 h |
6. Conclusion

In this paper, we present an analytical study for the most recent research that uses machine learning based on HPCs for malware detection. We found that the most features of HPCs used in the literature are branch instructions, instructions, branch_misses, and cache references. We also noticed that the J48, JRip, MLP, and OneR algorithms are used more frequently than others and the most effective ones regarding the average of accuracy results are CNN, BayesNet, MLP, and J48, respectively. Moreover, we have simulated three experiments from the literature, using NN-based algorithms including MLP, CNN, and Full Order RBF that achieved 96.95%, 98.22%, and 98.68% accuracy in the testing phase respectively.

For future work, we are planning to use the proposed MLP, CNN, and Full Order RBF algorithms with real HPCs dataset to measure the performance of them and compare the results with other approaches.

7. Acknowledgment

The authors would like to thank King Fahd University of Petroleum and Minerals for providing the support and facilities to perform this research work.

8. References

[1] Pattee, J., & Lee, B. K. (2020). Intelli-Dynamic Malware Detection Based on Processor Behaviors. In 17th International Conference on Information Technology–New Generations (ITNG 2020) (pp. 113-120). Springer, Cham.

[2] Sayadi, H., Makrani, H. M., Dinakarrao, S. M. P., Mohsenin, T., Sasan, A., Rafatirad, S., & Homayoun, H. (2019, March). 2smart: A two-stage machine learning-based approach for runtime specialized hardware-assisted malware detection. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 728-733). IEEE.

[3] Sayadi, H., Gao, Y., Makrani, H., Sasan, A., Lin, J., Rafatirad, S., & Homayoun, H. Towards Run-Time Hardware-Assisted Stealthy Malware Detection.

[4] Sayadi, H., Makrani, H. M., Randive, O., PD, S. M., Rafatirad, S., & Homayoun, H. (2018, August). Customized machine learning-based hardware-assisted malware detection in embedded devices. In 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (pp. 1685-1688). IEEE.
[5] Wang, X., Konstantinou, C., Maniatakos, M., & Karri, R. (2015, November). Confirm: Detecting firmware modifications in embedded systems using hardware performance counters. In 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 544-551). IEEE.

[6] Kuruvila, A. P., Kundu, S., & Basu, K. (2020, July). Analyzing the Efficiency of Machine Learning Classifiers in Hardware-Based Malware Detectors. In 2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (pp. 452-457). IEEE.

[7] Bazm, M. M., Sautereau, T., Lacoste, M., Sudholt, M., & Menaud, J. M. (2018, April). Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In 2018 Third International Conference on Fog and Mobile Edge Computing (FMEC) (pp. 7-12). IEEE.

[8] Sayadi, H., Patel, N., PD, S. M., Sasen, A., Rafatirad, S., & Homayoun, H. (2018, June). Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification. In 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

[9] Wang, X., & Karri, R. (2015). Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 35(3), 485-498.

[10] Das, S., Werner, J., Antonakakis, M., Polychronakis, M., & Monrose, F. (2019, May). SoK: The challenges, pitfalls, and perils of using hardware performance counters for security. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 20-38). IEEE.

[11] Krishnamurthy, P., Karri, R., & Khorrami, F. (2019). Anomaly detection in real-time multi-threaded processes using hardware performance counters. IEEE Transactions on Information Forensics and Security, 15, 666-680.

[12] Kuruvila, A. P., Kundu, S., & Basu, K. (2020). Defending Hardware-based Malware Detectors against Adversarial Attacks. arXiv preprint arXiv:2005.03644.

[13] Wang, X., Konstantinou, C., Maniatakos, M., Karri, R., Lee, S., Robison, P., ... & Kim, S. (2016). Malicious firmware detection with hardware performance counters. IEEE Transactions on Multi-Scale Computing Systems, 2(3), 160-173.

[14] Kwan, A. (2020). Malware Detection at the Microarchitecture Level using Machine Learning Techniques. arXiv preprint arXiv:2005.12019.

[15] Leng, E. W. L., Zwolinski, M., & Halak, B. (2017, July). Hardware performance counters for system reliability monitoring. In 2017 IEEE 2nd International Verification and Security Workshop (IVSW) (pp. 76-81). IEEE.

[16] Retrieved December 30, 2020, from https://colab.research.google.com/

[17] Retrieved December 30, 2020, from http://www.kfupm.edu.sa/centers/ite/default.aspx/

[18] R. Tahir, “A study on malware and malware detection techniques,”International Journal of Education and Management Engineering, vol. 8, no. 2, p. 20, 2018

[19] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, “A survey on heuristics malware detection techniques,” inInformation and Knowledge Technology (IKT),2013 5th Conference on. IEEE, 2013, pp. 113–120.

[20] I. A. Saeed, A. Selamat, and A. M. Abuagoub, “A survey on malware and mal-ware detection systems,”International Journal of Computer Applications, vol. 67, no. 16, 2013.