Regular Research Paper; An Embedded Malware Detection System Using a Support Vector Machine

Rob Oshana **(Contact Author)**
Edge Processing Business Line
NXP Semiconductors
Austin, USA
robert.oshana@nxp.com

Mitchell A. Thornton
Darwin Deason Institute for Cybersecurity
Southern Methodist University
Dallas, Texas, USA
mitch@smu.edu

Mike Caraman
Edge Processing Business Line
NXP Semiconductors
Bucharest, Romania
mike.caraman@nxp.com

Nathan Srirama
Southern Methodist University
Dallas, Texas USA
nsrirama@mail.smu.edu

***Abstract*—A prototype detection system for side channel attacks using a Support Vector Machine and processor performance counters is proposed. The performance and robustness of the approach is assessed. The system is capable of detecting multiple malicious exploits simultaneously. Dimensionality reduction techniques are used to improve model performance without losing accuracy. Feature selection techniques are used to determine new feature subsets. Effectiveness is measured using Receiver Operating Characteristics. The robustness of the detection system is measured using a Gaussian noise model and comparing the Root Mean Square Error and accuracy to the standard deviation of noise at different noise levels. e measure. We assess the robustness of the detection system to CPU load using CPU stress applications and CPU utilization limits.**

Keywords— Edge, embedded, machine learning, feature selection, event counter

I. Introduction

Computing at the edge of a network requires a broad range of capabilities to achieve optimal security, energy efficiency, connectivity, performance, and machine learning intelligence. These capabilities often times must work together to achieve desired goals. For example, protecting an IoT system from a malicious attack may require a combination of security, performance, and machine learning. Detecting and suppressing malicious attacks in IoT and Edge processing is an on-going challenge for researchers and vendors alike [1][2][3].

A method for the detection of malware using performance counters have been demonstrated and verified [4][5]. Additionally, embedded and edge-based techniques using support vector machines (SVM) have been devised and experimentally verified [6][7]. The techniques in [6] and [7] are based upon monitoring on-board hardware event counters and use a SVM for detection. The technique requires a minimal amount of modification to hosting computer systems since it uses pre-existing event counters and supporting circuitry and associated system software assets with no additional hardware required.

This paper describes the initial results and the evaluation approach of [6][7] under a variety of system stress levels to predict robustness of the method. ROC curves are provided for varying discrimination levels of the classifier, enhanced feature selection is applied, hyperparameter tuning is applied, and varying levels of noise were added to the side channels to determine the point at which the system would lose functionality.

II. Side Channel Attack Detection

The detection approach in [7] has shown effective results in detecting common variants of SPECTRE that are running in a system that is also executing multiple other non-threatening tasks.

The architecture of the detection system is shown in Fig. 1. The Embedded Detection Agent running on the embedded device uses “perf” to collect hardware-based event samples from the applications running on the target device and sends them to the Edge Detector requesting a classification. If the Edge Detector predicts that there is a malicious attack running on the embedded device process, it will kill or suspend the process. The embedded devices we target include ARM-based systems such as the iMX heterogeneous SoC as well as x86 based systems.

Machine learning training and classification is a component of the Edge Detector. Modifications were made to the Linux kernel to allow a simulated attack. Specifically, a victim kernel module, which is the SPECTRE module, designed to have a vulnerable driver, is implemented in the kernel and is used to demonstrate the malware attack against the kernel space. Additional kernel modifications were made to grant access to the Performance Measurement Unit (PMU) counters from user space since the SPECTRE proof of concept uses the PMU counters as a timing measurement method.

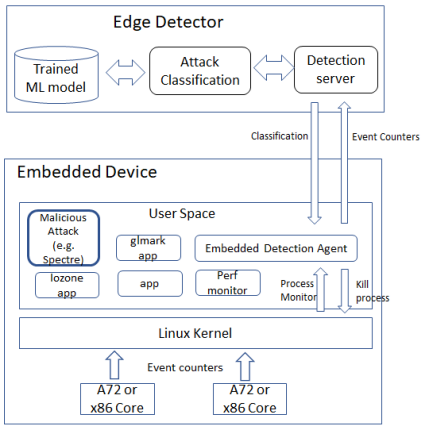


Fig. 1 Architecture for the detection system

The sequence diagram showing the interactions between the Edge Detector, Embedded Detection Agent, and the embedded processes running in user space is shown in Fig. 2. Our detection system continuously polls for top three CPU loaded processes for detection analysis in this prototype.

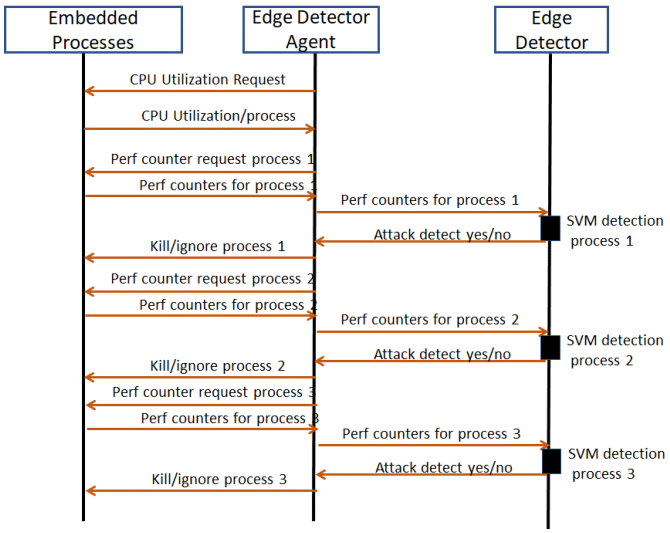


Fig. 2 Sequence Diagram for Edge Detection System

III. Previous Results

In [6] a SVM was used for detection of a simulated SPECTRE attack. The detection system has shown effective results in detecting a common variant of SPECTRE running in a system including multiple other non-threatening tasks including;

1. System idle task
2. I/O operations on the network file system
3. I/O operation on a sdcard using iozone file benchmarking tool
4. Graphics operations using an OpenGL benchmark called glmark2
5. TCP/IP communication using iperf measurement tool
6. I/O operations on the Linux networking file system using a tool called fio
7. Openssl crypto operations
8. The benchmarking application lmbench which measures latency and bandwidth
9. SPECTRE attack; the standard proof of concept SPECTRE attack

The work in [6] analyzed a variety of different machine learning algorithms. Fig. 3 shows the cross validation results of several machine learning classifiers including decision trees, gaussian naïve bayes, random forests, *K*-nearest neighbors support vector machines, and multilayer perceptrons.

Fig. 3 shows the mean *recall*, *precision* and *F*1-score averaged across all testing folds. All machine learning methods performed well, and two methods perform perfectly on the training and test sets: K Nearest Neighbor (*K*NN) and Support Vector Machine (SVM). SVM was chosen over *K*NN because of the time it takes to predict when the model is deployed.

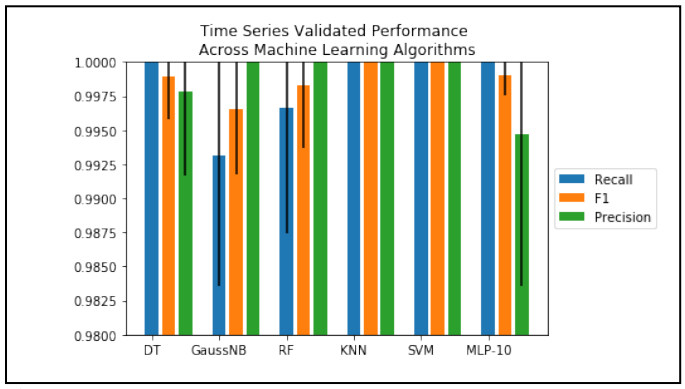


Fig. 3; Time series validated performance of machine learning models based on K-fold cross validation

The results from [6] showed true positive rates in excess of 98% with corresponding false positive rates less than 1%. In many cases, a 0% false positive rate is achieved. This work did not consider the incorporation of feature selection nor did it evaluate performance over a range of system stress factors and operating points.

In [7] the system in [6] was expanded to show detection of multiple variants of malicious attack with encouraging results. The results are summarized in Table I which represents SVM performance for the initial SPECTRE prediction. Table II shows SVM performance prediction for multiple variants of malware attack including the original SPECTRE variant, a micro-ops cache based variant [8], a Chrome browser variant [9] and a Spook.js malware variant [10]. The results are consistent with the SPECTRE performance presented in Table I. The detection system was able to detect all malware variants simultaneously in the presence of other application processes executing in parallel.

TABLE I. SVM PERFOMANCE SPECTRE

	Precision	Recall	F1 Score	Support
0	0.999	0.997	0.998	19555
1	0.963	0.990	0.976	1381

TABLE II. SVM PERFORMANCE ALL ATTACKS

	Precision	Recall	F1 Score	Support
0	0.999	0.996	0.998	19551
1	0.960	0.987	0.974	1718

IV. ROBUSTNESS AND PERFORMANCE OF MALWARE DETECTION SYSTEM

A key part of this detection system method is feature selection which includes the identification of the relevant event counters to extract from the system for inferencing and detection. The selected events should contain counters that provide relevant information to indicate side-channel attack operations and their side effects. In [6] the event counters were selected by subject matter experts. A smaller number of event counters were selected that met the hardware extraction constraints for each processor. In [7] the performance counter selection based on a more rigorous feature selection approach.

A. Performance Counters Constraints

The hardware event counters, which are a core component to this solution, are limited resources. For example, the i7-6950X x86 Broadwell processor used in previous investigation comprises 5 physical counters in the Performance Monitoring Unit, while the i7-6600U x86 Skylake processor selected for this investigation comprises only 4 counters. ARM Cortex-A72 cores present in i.MX8 processors comprises 6 counters. This limits how much information can be extracted in each cycle from the processor.

New enhancements to modern kernels and the perf tool provide software mechanisms for counter multiplexing and cycle scaling. Using cycle normalization allows for an extended list of events to be captured with high precision. perf provides a grouping mechanism for events which improves the precision even further. This is accomplished by sampling the cycle counters for each sub-group.

Our approach focused on feature selection methods from a larger set of counters which capture side channel attack operations and related side effects. Key categories include speculative execution, branch prediction and cache operations In particular, for x86 Skylake we identified 35 relevant counter types. On Arm A72 we identified 38 relevant counter types that match these criteria.

B. PCA Dimensionality Reduction

Dimensionality reduction techniques can be used to reduce the number of features in our dataset without losing accuracy and information so that its possible to preserve or improve the model performance.

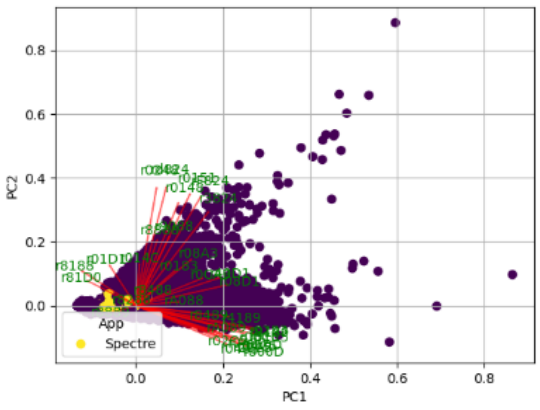
PCA is a dimensionality reduction approach that focuses on feature extraction. PCA can compress a dataset into a lower dimensional feature subspace with the principal goal of maintaining most of the relevant data [11][12]. Here we use PCA to determine which features are important for best describing the variance in the data set.

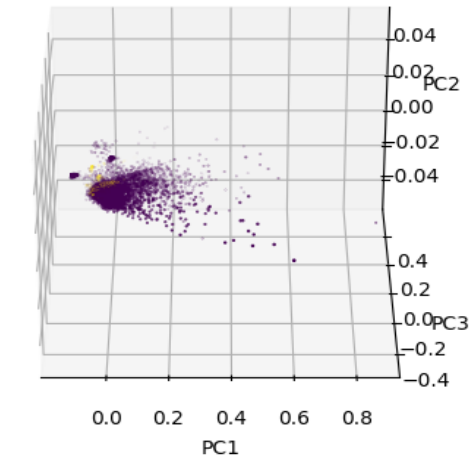
Table III shows a summary of the results of PCA for the x86 and Arm based systems. Thirty five x86 counters were collected for the SPECTRE based attacks. The top twelve principal components provide 90% variance on the data. A similar number of Cortex-A72 counters were collected for the same attacks. For Cortex-A72, the top ten principal components provide 90% variance on the data. Our cumulative variance target was 90%.

TABLE III. PCA RESULTS FOR X86 AND ARM CORTEX-A72 PERFORMANCE COUNTER DATA. PRINCIPLE COMPONENTS AND FEATURE CONTRIBUTIONS

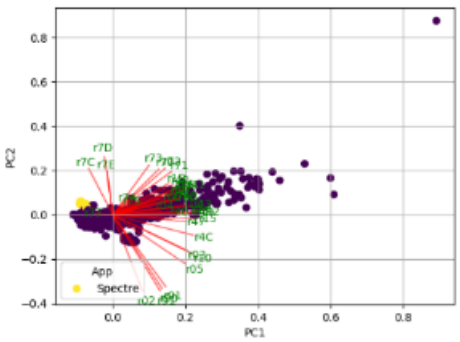
Principle Component	Variance x86	Variance Arm A72
1	0.33	0.35
2	0.16	0.15
3	0.11	0.11
4	0.07	0.10
5	0.05	0.05
6	0.04	0.05
7	0.03	0.04
8	0.03	0.03
9	0.03	0.02
10	0.02	0.02
11	0.02	
12	0.02	
Total	0.903	0.92

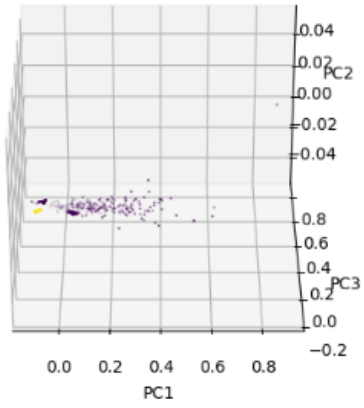
In Fig. 4 shows the transformed counter data in the planar space of the top two principal components for x86 and Cortex-A72. The counter contribution to these principal components is highlighted with the red arrows representing the eigen vectors. Fig. 4 also shows the transformed counters data into the 3D space of the top three principal components and highlights the variance and separation of the data in the new sub-space.





a. x86





b. Cortex A72

Fig. 4 : 2D and 3D representation of the top two principal components with the contribution of each performance counter data for SPECTRE attack variant for a. x86 and b. Cortex-A72

PCA has shown that a subset of the event counters we chose can we reduced but still achieve at least 90% variance of the data. But we still may have missed an important feature in our manual selection process. In order to assess this, we use algorithmic techniques search for missing features to improve detection system performance.

C. *Feature selection optimization*

While PCA attempts to reduce dimensionality by exploring how one feature of the data is expressed in terms of the other features (linear dependency), feature selection is a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. This can reduce computational cost as potentially improve the performance of the model [13][14]. In our case this can also be used to adhere to the constraints on the hardware available for extracting event counters from the processor. This is done by selecting a subset of core events that are most effective in prediction accuracy.

We used the LASSO feature selection functions from the scikit library to determine feature importance. LASSO (Least Absolute Shrinkage and Selection Operator) is a statistical formula with the main purpose of feature selection and regularization of the data model [15]. In our detector, the features (input variables to the model) are the core events. Choosing the right input variables improves the accuracy of our model. The features selection phase of LASSO helps in the proper selection of these variables.

The results of the feature selection for x86 and Arm are listed in and Table IV and Table V.

TABLE IV: KEY X86 CORE EVENTS

Event count	Event description
r0248	Number of times a request needed a FB entry but there was no entry available for it
r0480	Cycles where a code fetch is stalled due to L1 instruction cache miss.
r0CA3	Execution stalls while L1 cache miss demand load is outstanding
r40D1	Retired load instructions which data sources were load missed L1 but hit FB due to preceding miss to the same cache line with data not ready
r8889	Taken speculative and retired mis-predicted indirect branches with return mnemonic
r0283	Instruction fetch tag lookups that miss in the instruction cache (L1I)
r3824	Requests from the L1/L2/L3 hardware prefetchers or Load software prefetches that miss L2 cache
r010D	Core cycles the allocator was stalled due to recovery from earlier clear event
r01C5	Mis-predicted conditional branch instructions retired
r8189	Taken speculative and retired mis-predicted macro conditional branches
rF824	Requests from L2 hardware prefetchers

TABLE V: KEY A72 CORE EVENTS

Event count	Event description
r7C	Barrier speculatively executed - ISB
r7D	Barrier speculatively executed - DSB
r7E	Barrier speculatively executed - DMB
r1B	Operation speculatively executed
r12	Predictable branch speculatively executed
r4C	Level 1 data TLB refill - Read
r75	Operation speculatively executed - VFP

We integrated the performance counter events in Tables IV and V into our detection system and did a comparison of the SVM performance results. Table VI shows detector performance for negative results (normal applications) comparing the same sets of event counters discussed previously:

- Row "6 event counters" - the baseline manual selection of event counters used in [6].
- Row "35 event counters" - all 35 relevant event counters used in detection, prior to feature selection.
- Row "11 principal components" - the key principal components accounting for 90% variation.
- Row "11 features selection" – a reduced set of 11 event counters selected from the superset of 35 event counters using embedded approaches to feature selection, specifically Lasso and Gradient Boost feature selection methods from scikit
- Row "5 features selection" - the top 5 event counters selected from the superset of 35 event counters using embedded approaches to feature selection, specifically Lasso and Gradient Boost feature selection methods from scikit

Similarly, Table VII shows SVM performance for positive results (attacks) comparing the same event counter (feature) options.

The comparison from Table VI and Table VII shows that the new list of event counters proposed in this study, based on principal component analysis and feature selection algorithms, provides significantly better results than [7] and with similar performance to the experiment with the larger number of counters.

TABLE VI. SVM PERFORMANCE COMPARISON ON X86 – NEGATIVE RESULTS

Features/ Negative results	Precision	Recall	F1 Score	Support
6 event counters from [6]	0.987	0.982	0.984	7381
35 event counters	1.000	1.000	1.000	7347
11 principal components	1.000	1.000	1.000	7353
11 features selection	1.000	1.000	1.000	7393
5 features selection	1.000	0.999	0.999	7291

TABLE VII. SVM PERFORMANCE COMPARISON ON X86 – POSITIVE RESULTS

Features/ Negative results	Precision	Recall	F1 Score	Support
6 event counters from [6]	0.978	0.983	0.981	5922
35 event counters	1.000	1.000	1.000	5956
11 principal comp	1.000	1.000	1.000	5950
11 features selection	1.000	1.000	1.000	5910
5 features selection	0.999	1.000	0.999	6012

V. RECEIVER OPERATING CHARACTERISTIC

We diagnose the effectiveness of our detection system using a receiver operating characteristic (ROC) curve which plots the true positive rate (TPR) against the false positive rate (FPR) at various discrimination threshold settings [16][17]. We generated the ROC by re-combining and re-splitting the collected event counter data randomly.

The TPR defines how many correct positive results occur among all positive samples available during the test. FPR, on the other hand, defines how many incorrect positive results occur among all negative samples available during the test.

$$\text{True Positive Rate} = \text{Sensitivity} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

= Probability of detection

$$\text{False Positive Rate} = (1 - \text{Sensitivity}) = \frac{False\ Positives}{False\ Positives + True\ Negatives}$$

= Probability of false alarm

Fig. 5 shows the ROC for the detection system. Classifiers that produce curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal where FPR = TPR. The ROC in Fig. 5 indicates that our detection system is performing well.

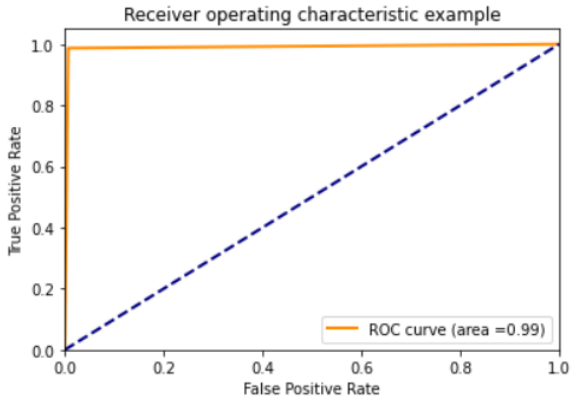


Fig. 5. ROC for Detection System

The Area Under the Curve (AUC) aggregates the performance of the model at all threshold values and performs well as a general measure of predictive accuracy [18]. The best possible value of AUC is 1 which indicates a perfect classifier. The AUC for our detection system is 0.99.

VII. HYPERPARAMETER OPTIMIZATION

Hyperparameter optimization was used to assess additional performance improvements for our detection system [19][20]. We used a Grid Search algorithm available in scikit to spot check the data by defining a search space as a bounded domain of hyperparameter values and randomly sampling points in that domain.

The hyperparameter tuned SVM performed exactly the same as the default on the testing dataset. Table VIII shows a summary of the hyperparameter search. The "mean_test_score" and "std_test_score" are the portion of the training dataset that was used during cross-validation during training. In total, 180 models were tested. The "mean" and "std" rows are the mean and standard deviation of all 180 models' performances.

Likewise, min, max, and the percentages refer to quartiles. The min and max were the minimum and maximum values found from the models, and the percentages rows are the values of models found at the respective quartiles. The hyper-tuned SVM has more false-negative predictions as shown in the confusion matrix Fig. 6. As the amount of training data is increased, the hyper-tuned SVM records fewer false negatives. Additionally, as the amount of training data is increased, there are more false negatives the default-SVM records. This difference is small, just 1-2 false negative values in total, since there were already very few false negatives to begin with. The hyper-tuned SVM is less prone to overfitting as the size of input data increases, which we believe is related to the lower standard deviation of the training validation scores. The hyper-tuned model parameters were found to have lower standard deviation on their testing scores than the default parameters.

TABLE VIII: GRID SEARCH MEAN RESULTS

METRIC	MEAN TEST SCORE	STD TEST SCORE
Count	180	180
Mean	0.949039	0.001029
Std	0.113166	0.001531
Min	0.551556	0.000000
25%	0.970107	0.000066
50%	0.987477	0.000337
75%	0.999081	0.001228
Max	0.999963	0.006937

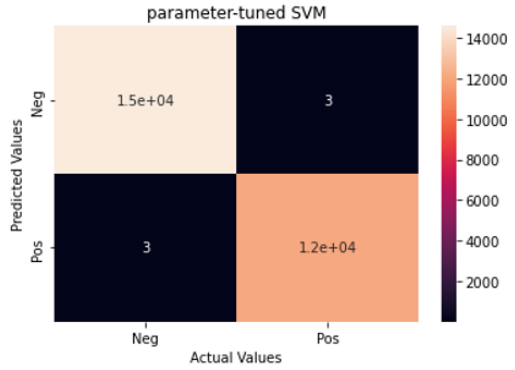


Fig. 6 Hyper-Tuned SVM Confusion Matrix

VIII. GAUSSIAN NOISE ANALYSIS

We assess the robustness of our detection system using different level of "noisy" inputs and analyze the impact on the confusion matrix results. We used the normal distribution function from the "NumPy" (numerical python) package to generate the median and the mean of the data [21].

We repeatedly rerun the trained classifier using test data with offsets generated from a standard distribution with increased values of standard deviation with a mean of zero. The noise standard deviation values ranges from 1 to 70% of the test data standard deviation. Fig. 7 plots the TP, TN, FP, and FN results over the standard deviation of noise percentage. The results show the detection system behaving well in the presence of noise with a standard deviation up to 35% of the original testing data.

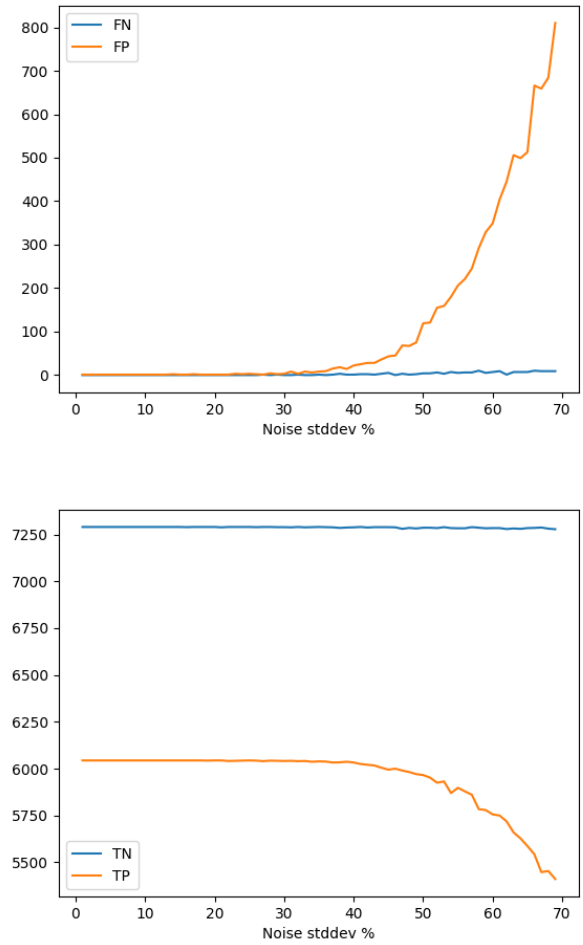


Fig. 7 Confusion Matrix results for gaussian noise with different standard deviations

IX.CPU LOAD ANALYSIS

We analyze the robustness of our detection system to CPU load using CPU stress applications and CPU utilization limits. We use the "tasket" Linux tool to set the CPU affinity for the running attack process using its Process ID (pid) as well as the stress applications to specific cores used for jamming. In the first experiment the Linux scheduler assures CPU fairness and divides the cycles between the processes (33% to attack and 33% to each of the two CPU stress applications). We tested multiple CPU stress applications such as YouTube, a subset of the stress applications described in [7], and other custom applications. In a second experiment we limited the CPU cycles when the attack process was running alone without stress using the "cpulimit" Linux tool which is used to limit the CPU usage of a process.

In both scenarios, no change in detection system performance was observed. The CPU load has no negative

impact on the detection system because we normalize the side channel counter information by the number of cycles.

X. Conclusion

We expand on a method for the detection of branch prediction and speculative side channel attacks using hardware performance counters and a support vector machine. The technique is based upon monitoring on-board, hardware event counters rather than characteristics of the targeted data. The technique requires a minimal amount of modification to an edge-based computer system since it uses pre-existing event counters and supporting circuitry and associated system software assets with no additional hardware required. Multiple variants of the attack were reproduced and detected concurrently including a standard SPECTRE variant, a micro-ops cache based variant, a Chrome browser variant and a Spook.js malware variant.

A more robust method for selecting event counters was investigated and validated against the original SPECTRE attack. Feature selection algorithms were used to determine the optimum event counters to achieve maximum performance. The performance of the new counter selections was compared against the original selection based on subject matter expert analysis and showed significantly better results.

We measured the effectiveness of our detection system to distinguish between classes using ROC and AUC and determined that the detection system performs robustly.

We assessed speed and quality of our detection system learning process using hyperparameter optimization and concluded that our detection system was optimized.

We assessed the robustness of the detection system by executing experiments with increasing amounts of application noise using a gaussian model with a scalable selection of random noise as well as experiments in varying the CPU loading of the stress applications and attack processes.

References

[1] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L. Titzer, Toon Verwaest, Spectre is here to stay An analysis of side-channels and speculative execution, Feb 2019,

[2] Agarwal, A., O'Connell, S., Kim, J., Yehezkel, S., Genkin, D., Ronen, E., and Yarom, Y., "Spook.js: Attacking Chrome strict site isolation via speculative execution"

[3] Ren, X., Moody, L., Taram, M., Tullsen, D., Jordan, M., and Venkat, A., "I see dead μops: leaking secrets via Intel/AMD micro-op caches", International Symposium on Computer Architecture (ISCA), 2021.

[4] L. Congmiao, J. Gaudiot, "Detecting Spectre attacks using hardware performance counters", IEEE Transactions on Computers, May 2021

[5] John Demme Matthew Maycock Jared Schmitz Adrian Tang Adam Waksman Simha Sethumadhavan Salvatore Stolfo, "On the Feasibility of Online Malware Detection with Performance Counters", ACM SIGARCH Computer Architecture NewsVolume 41Issue 3June 2013 pp 559–570

[6] R. Oshana, M. Thornton, X. Roumegue, E. Larson, "Real-time edge processing detection of malicious attacks using machine learning and processor core events", 15th Annual IEEE International Systems Conference, April 15, 2021.

[7] R.Oshana, M. Thornton, M. Caraman, "A side channel attack detection system using processor core events and a support vector machine" 11th Mediterranean Conference on Embedded Computing, 2022.

[8] Ren, X., Moody, L., Taram, M., Tullsen, D., Jordan, M., and Venkat, A., "I see dead μops: leaking secrets via Intel/AMD micro-op caches", International Symposium on Computer Architecture (ISCA), 2021, pp 361-374

[9] Agarwal, A., O'Connell, S., Kim, J., Yehezkel, S., Genkin, D., Ronen, E., and Yarom, Y., "Spook.js: Attacking Chrome strict site isolation via speculative execution", https://www.spookjs.com/files/spook-js.pdf

[10] S. Rottger and Artur Janc, "A Spectre proof-of-concept for a Spectre proof web",https://security.googleblog.com/2021/03/a-spectre-proof-of-concept-for-spectre.html, March 2021

[11] Reddy G., et al "Analysis of dimensionality deduction techniques on big data", IEEE Xplore, March 16, 2020

[12] Han, X., Xu, L., and Ren, M., "A naive bayesian network intrusion detection algorithm based on principal component analysis", 2015 7th International Conference on Information Technology in Medicine and Education

[13] Anam Fatina, Ritesh Maurya, Malay Dutta, Radim Burget, and Jan Masek, "Android malware detection using genetic algorithm based optimized feature selection and machine learning", 42nd International Conference on Telecommunications and Signal Processing", 2019

[14] P Illavarason and B Kamachi Sundaram, "A study of intrusion detection system using machine learning classification algorithm based on different feature selection approach", Third International Conference on I-SMAC, 2019

[15] R Muthukrishnan and R Rohini, "LASSO: A feature selection technique in predictive modeling for machine learning", 2016 IEEE International Conference on Advances In Computer Applications.

[16] Wang Xu-Hui, Shu Ping, Cao Li, and Wang Ye, "A ROC curve method for performance evaluation of support vector machine with optimization strategy", 2009 International Forum on Computer Science-Technology and Applications, December 2009

[17] Ronaldo Cristiano Prati, Gustavo Batista, and Maria Carolina Monard, "Evaluating classifiers using ROC curves", IEEE Latin America Transactions, August 2008

[18] Jin Huang and C.X. Lin, "Using AUC and accuracy in evaluating learning algorithms", IEEE Transactions on Knowledge and Data Engineering, 2005

[19] Abdullah Ammar Karcioglu and Hasan Bulut, "Performance evaluation of classification algorithms using hyperparameter optimization", 6th Annual International Conference on Computer Science and Engineering, 2021

[20] Hussain Alibrahim and Simone A. Ludwig, "Hyperparameter optimization: comparing genetic algorithm against grid search and baysian optimization", IEEE Congress on Evolutionary Computation, 2021

[21] Sophia Susan Raju, Boyan Wang, Kashyap Mehta, Ming Xiao, "Application of noise to avoid overfitting in TCAD augmented machine", 2020 International Conference on Simulation of Semiconductor Processes and Devices, 2020