A Side Channel Attack Detection System Using Processor Core Events and a Support Vector Machine

Rob Oshana
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
mailto:mitch@smu.edu

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

***Abstract*—A method for the detection and suppression of side band channel attacks has been proposed and evaluated using machine learning and processor core events. A supervised learning model is used in the implementation of a system based on hardware event counters to detect malicious exploits such as SPECTRE variants running in a process on a Linux based system operating as an Edge computing device. The approach uses existing on-chip hardware to detect variants of malicious exploitation in the midst of other application processes and suspend the offending process. In this work we analyze multiple variants of side channel attack and demonstrate how our detection system can be trained to detect and react to multiple attacks simultaneously. We demonstrate this prototype on variants of the classic SPECTRE attack such as the micro-ops cache attack based on x86 based machines. We use dimensionality reduction techniques and feature selection techniques from a large set of counter data to improve performance results. The detection system has successfully performed on multiple instruction set architectures including x86 as well as Cortex-A class ARM architectures.**

***Keywords*— Edge Processing, supervised learning, malicious event, SPECTRE, embedded system, event counter, SVM, micro-ops cache, feature selection**

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 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 battle for researchers and vendors alike.

One of the more famous side channel attacks is SPECTRE, a security vulnerability that exploits speculative execution and indirect branch prediction circuitry that is common and present in most modern CPU cores. The exploit allows access to unauthorized information by implementing side channel analysis of information in the data cache of the system [1]. SPECTRE is documented in the Common Vulnerabilities and Exposures (CVE) database as CVE-2017-5717 and CVE-2017-5753. The general idea behind the attack is that the attacker uses the performance enhancement features of the processor, namely the cache and branch predictor plus speculative execution circuitry, to read higher privileged data.

A method for the detection of SPECTRE using performance counters have been experimentally verified [2]. Additionally, edge based techniques using support vector machines (SVM) have been devised and experimentally verified [3]. The technique in [3] 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 hosting computer systems since it uses pre-existing event counters and supporting circuitry and associated system software assets with no additional hardware required. The disclosed SPECTRE detection technique has been experimentally shown to be effective for Linux based operating systems in a real time embedded system based on a prototype implementation and associated experimental results.

Variants of SPECTRE continue to be discovered. These variants use methods such as bounds check bypass store (CVE-2018-3693), branch target injection (CVE-2017-5715), speculative store bypass (CVE-2018-3639), and same exception level training (CVE-2022-23960).

Other SPECTRE variants exploit the micro-op cache on x86 machines. A micro-op cache speeds up computing by storing simple commands which allows the processor to fetch these commands quickly and earlier in the speculative execution process. Hackers can steal data when a processor fetches commands from the micro-op cache.

The goal of this paper is to design a generic detection system to reliably detect and suppress multiple variants of the SPECTRE class of side channel attack in the presence of other applications executing in parallel.

We will assess the efficacy of the detection system using the key metrics of precision, recall, and F1. Precision measures how many of the positive predictions our detector makes correctly and reflects the ability of the detector not to label as positive (attack) a sample that is in fact negative (no attack). Recall measures how many of the positive cases our detector correctly detected over all of the positive cases in the data and measures the ability of the detector to find all positive samples (attacks). F1 is a metric that weights both Precision and Recall in a balanced way. A higher F1 score is achieved when both Precision and Recall are high.

II. Previous Results

The detection system in [3] has shown effective results in detecting a common variant of SPECTRE running in a system including multiple other non-threatening tasks (1-8);

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 [3] analyzed a variety of different machine learning algorithms. Fig. 1 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.

The bar chart in Fig. 1 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.

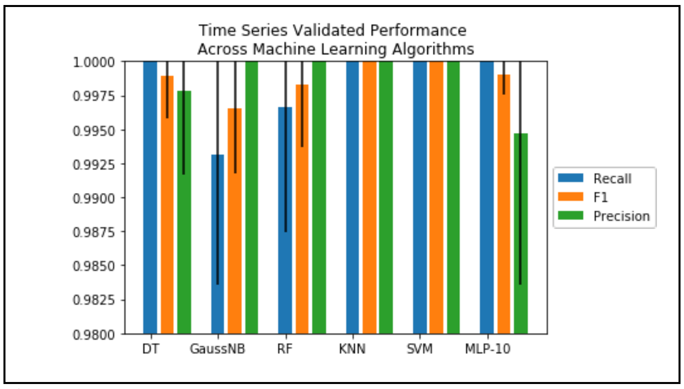


Fig. 1; Time series validated performance of machine learning models based on K-fold cross validation

III. Generic Side Channel Attack Detection

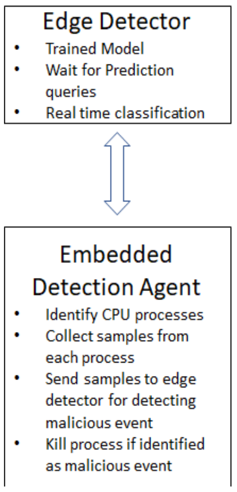
Our work significantly enhances the detection capabilities of that demonstrated in [3]. Our work demonstrates the ability to detect multiple variants of malicious attack simultaneously in with similar suppression capability. Our detection system operates robustly in the presence of varying amounts of application "noise".

A. *Prototype Detection System*

Our work expands upon the prototype architecture and results in [3];

1. Creating a generic detector that can operate as an Edge processing system as well as cloud based.
2. Detecting multiple variants of the SPECTRE attack and other forms of malicious attack such as that described in [4] simultaneously using an enhanced machine learning model.
3. Enhanced performance counter analysis using dimensionality reduction techniques such as Principal Component Analysis (PCA) to select an orthogonal and dominating set of events to be counted to further enhance detection performance for the exploits previously studied and the new exploits considered here.
4. Effective detection on both ARM and x86 instruction set architectures.

Fig. 2 shows the two main components of the prototype implementation. The Embedded Detection Agent collects event counter samples that represent the runtime profiles of the active processes on the embedded system. In our prototype system, these samples are communicated to a computer representing an edge-resident processor and referred to as the "Edge Detector." The Edge Detector predicts whether or not the exploit is present. If the Edge Detector predicts that a malicious event is present, the process is killed by the "Embedded Detection Agent."



Edge Detector
Trained Model
Wait for Prediction queries
Real time classification
Embedded Detection Agent
Identify CPU processes
Collect samples from each process
Send samples to edge detector for detecting malicious event
Kill process if identified as malicious event

Fig. 2 Edge Detector and Embedded Detector Agent Components of the System Architecture

B. *Real-time Software Architecture*

The software architecture for the detection system is shown in Fig. 3. 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 as well. We made modifications to the Linux kernel to allow a simulated attack. Specifically, a SPECTRE victim kernel module, which is the SPECTRE gadget module designed to have a vulnerable driver, is implemented in the kernel and is used to demonstrate the SPECTRE 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.

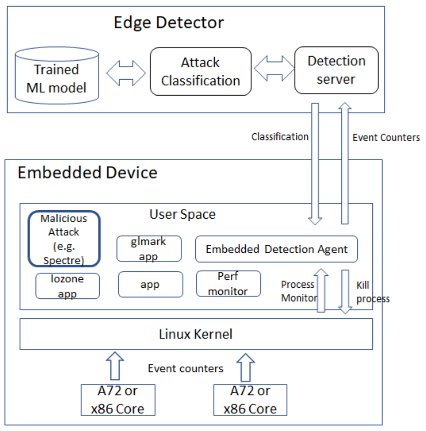


Fig. 3 Software Architecture for the detection system

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. 4. Our detection system continuously polls for top three CPU loaded processes for detection analysis in this prototype.

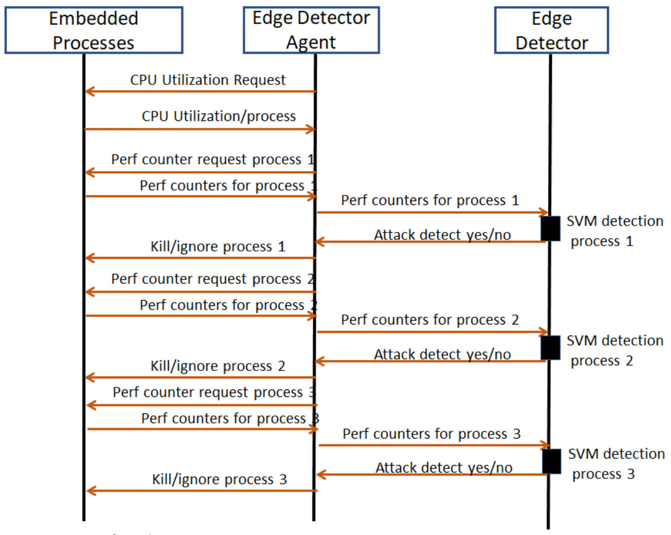


Fig. 4 Sequence Diagram for Edge Detection System

IV. Detection of Multiple Malicious Attacks

Successful detection of the original SPECTRE attack by our edge detector agent was encouraging, and motivated us to experiment with the detection of other variants of side channel attacks. We experimented with several newer forms of malicious attacks;

- Micro-op cache attack detection on x86 [4]
- Google SPECTRE browser proof of concept attack [5][6]
- Spook.js attack [7]

A. *Detection of New Attack Variants*

In order to assess the detection of the new attack variants we have replicated the setup published in [3] using the following core events (x86) to train a model and using SVM inferencing:

- L1 data cache loads (r81D0)
- L1 data line replacements (r0151)
- Not taken macro-conditional branches (r4189)
- Not taken speculative and retired mis-predicted macro conditional branches (r8188)

For each attack variant we created or obtained code that simulated the attack and ported that to our detection system as a process running in Linux user space.
The sequence of steps used to create and detect the attack variants is:

1. Replicate and integrate the attack code into the detection system environment
2. Collect the training dataset using the counters listed above for each attack variant, label as attack and non-attack, and integrate into a comprehensive attack model
3. Create an SVM inference
4. Use the dataset to train and test the SVM
5. Run the attack(s) in the presence of the other application processes
6. Collect counters from the top 3 processes
7. Use SVM inferencing to predict attacks in the presence of other application processes
8. Kill the attack process if the prediction is "attack"

B. *Attack Variant 1; Micro-op Cache*

A micro-op cache speeds up computing by storing simple commands which allows the processor to fetch these commands quickly and earlier in the speculative execution process. Hackers can steal data when a processor fetches commands from the micro-op cache. Reference [4] presents a detailed characterization of the micro-op cache in Intel Skylake and AMD Zen microarchitectures. Attacks can exploit the micro-op cache to leak secrets in three primary settings: (1) across the user-kernel boundary, (2) across co-located SMT threads running on the same physical core, but different logical cores, and (3) two transient execution attack variants that exploit the micro-op cache timing channel, bypassing many recently proposed defenses in the literature.

Through close collaboration with University of Virginia, we obtained a code implementation of the micro-ops attack variant and integrated it into our detection system environment.

Table I shows detector performance for the Micro-Ops attack variant. The results indicate an effective detector for the micro-op attack variant. Detection performance is effective with no false negative or false positive conditions. The Support column in Table I indicates the number of occurrences in each class

TABLE I: DETECTION PERFORMANCE FOR MICRO-OP ATTACK

	PRECISION	RECALL	F1-SCORE	SUPPORT
0	1.000	1.000	1.000	19571
1	1.000	1.000	1.000	218

C. *Attack Variant 2; Google SPECTRE Proof of Concept*

Web browsers are not immune to SPECTRE like attacks. Although web browser vendors have been collaborating on approaches intended to mitigate this form of attack, it still remains a concern and has led to web developers deploying various forms of application level mitigations. Google recently shared the results of their Google Security Team's research on the exploitability of SPECTRE against web users. The Google Security Team has developed a fast versatile proof-of-concept written in JavaScript which is designed to leak information from the browser's memory. The team also confirmed that this proof-of-concept, and its variants, function across a variety of operating systems, processor architectures, and hardware generations.

The proof of concept is publicly available in both the leaky.page web address and in the community github. We reproduced this attack using Chrome browser version 90.0.4430.85. We have tested it using our Edge Detector, performing a similar process as we did for the micro-ops attack.

Table II shows detection performance for the Google SPECTRE attack variant. These results indicate effective detection performance for this attack variant.

Table II: DETECTION PERFORMANCE FOR GOOGLE SPECTRE ATTACK

	PRECISION	RECALL	F1-SCORE	SUPPORT
0	1.000	1.000	1.000	19567
1	0.968	0.948	0.958	96

D. *Attack Variant 3; Spook.js*

Spook.js is a transient execution side channel attack which targets the Chrome web browser. Spook.js attacks exploit the SPECTRE vulnerability by taking advantage of the fact that web pages contain large amounts of program code that get executed on a user's computer each time it is loaded. When an infected page loads the code gets executed by the browser which enables the hacker to steal confidential data. This has been demonstrated against applications such as Tumblr, Lastpass, and a Google server [5].

Using the Spook.js proof of concept code, we created a similar environment to reproduce the attack using our detection system.

Table III shows detector performance for the Spook.js attack variant. In this experiment a larger number of false positives were detected. We believe this is because Spook.js is implemented as a single process executing inside a browser which contains both attack and "normal" application code. This makes it difficult to label the counters for the normal code properly. Nevertheless, the attack variant is still detected with a F1 score of .995.

Table III: DETECTION PERFORMANCE SPOOK.JS

	PRECISION	RECALL	F1-SCORE	SUPPORT
0	0.997	0.993	0.995	19589
1	0.938	0.971	0.955	2261

E. *Simultaneous Detection Of Attacks*

After confirming that we could detect the new attack variants separately, we extended the detection system to detect multiple types of attacks simultaneously, using one SVM model and training it with the combined recorded samples from the different attack variants.

The results listed in Table IV represent SVM performance for the initial SPECTRE prediction.

Table IV: 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 V: SVM PERFORMANCE ALL ATTACKS

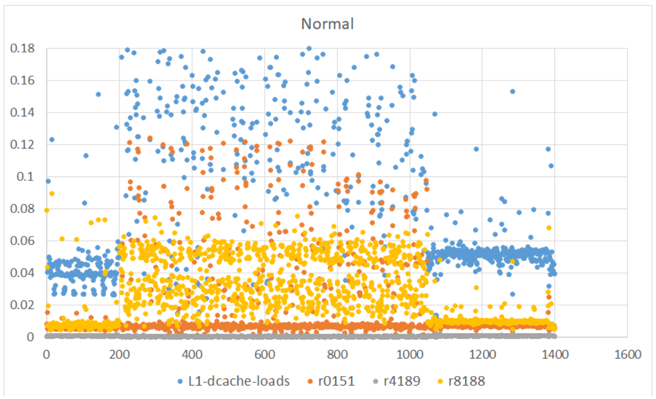
	PRECISION	RECALL	F1-SCORE	SUPPORT
0	0.999	0.996	0.998	19551
1	0.960	0.987	0.974	1718

Table V shows SVM performance prediction for the combined attacks. The results are consistent with the SPECTRE performance presented in Table IV..

F. *Performance Counter Analysis*

To better understand the behavior of the attacks and the success rate of our method, we performed an analysis of the time series representing the four selected core events (r81D0, r0151, r4189, r8188).

Fig. 5 shows a distribution profile of the counter events for non-attack applications versus the SPECTRE attack. The SPECTRE profile demonstrates a constant pattern with small standard deviation for each core event.



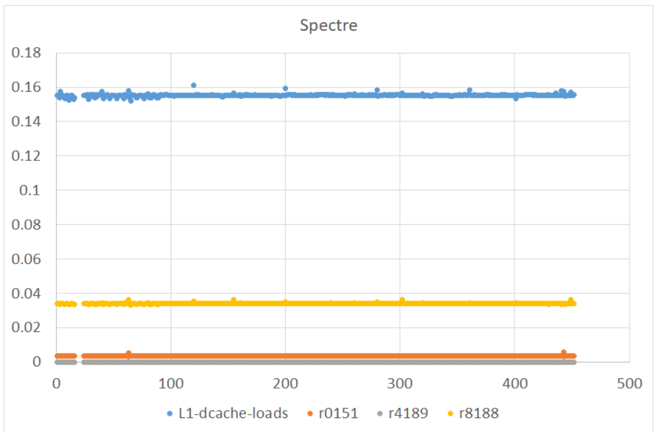


Fig. 5 : Distribution of counter values over time for the non-attacked application environment versus the distribution profile of a SPECTRE attack

Table VI: Performance counter profiles

MEAN	NORMAL	SPECT	MICROOPS	SPOOK
r81D0	0.169378896	0.1560	0.051453166	0.201235
r0151	0.016311552	0.0034	0.002399916	0.01737
r4189	0.00048171	0.0002	7.34789E-05	0.000281
r8188	0.169378896	0.0342	0.000715383	0.020877

STD DEV	NORMAL	SPECT	MICROOPS	SPOOK
r81D0	0.127270103	0.00553	0.00129856	0.083282
r0151	0.025319842	0.00061	2.4931E-05	0.009317
r4189	0.000276354	0.00004	1.1448E-05	0.000404
r8188	0.127270103	0.00174	0.00013805	0.024806

Table VI shows the attack profiles for the SPECTRE attack, the micro-ops attack variant and the Spook.js attack variant. Comparing this with the normal applications profile, the SPECTRE and Micro-ops profiles have different average and lower standard deviations. Alternatively, the Spook.js attack profile shows a larger standard deviation with significant overlap with the normal applications. This makes detection of this particular variant more challenging. We believe this is partially due to the way the attack simulation is implemented.

V. Performance Counter Selection

A critical part of our 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. The initial investigation was based on event counters that were selected by subject matter experts. We also selected a smaller number of event counters that met the hardware extraction constraints for each processor. Now we assess performance counter selection based on a more rigorous approach.

A. Extended Performance Counters

The hardware event counters, which are at the heart of this solution, are limited resources. For example, i7-6950X x86 Broadwell processor used in previous investigation comprises 5 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 side-effect. Key categories include speculative execution, branch prediction and cache operations In particular, for x86 Skylake we identified 35 counters types. On Arm A72 we identified 38 counters 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/information so that we preserve or improve the model performance. Dimensionality reduction has a number of benefits including reduced space requirements for storing data, less computation due to fewer dimensions, removal of redundant features which eliminates multicollinearity, removal of noise, and better visualization of the data among others [8].

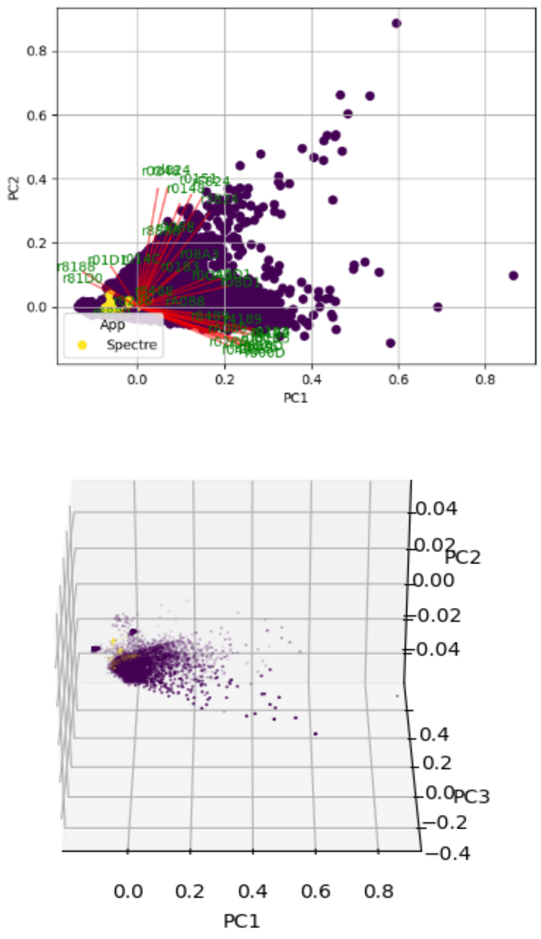
PCA is a dimensionality reduction approach that focuses on feature extraction. PCA can compress a dataset into a lower dimensional feature subspace with the principal goal of maintaining most of the relevant data [9]. Here we use PCA to determine which features are important for best describing the variance in the data set.

Table VII 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. Thirty eight Cortex-A72 counters were collected for the SPECTRE based attacks. The top ten principal components provide 90% variance on the data. 90% cumulative variance was our target.

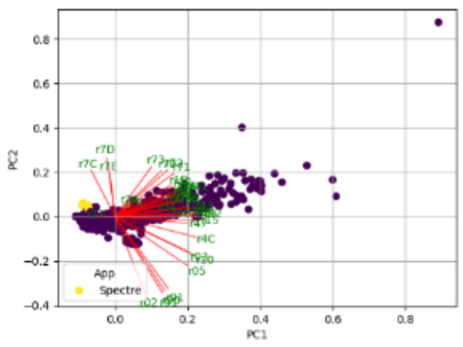
In Fig. 6, we have plotted the transformed counter data in the planar space of the top two principal components for x86 and Cortex-A72. We highlighted counter contribution to these principal components with the red arrows representing the eigen vectors. Fig. 5 also shows the transformed counters data into the 3D space of the top three principal components and highlights the variance and separation of the data in the new sub-space.

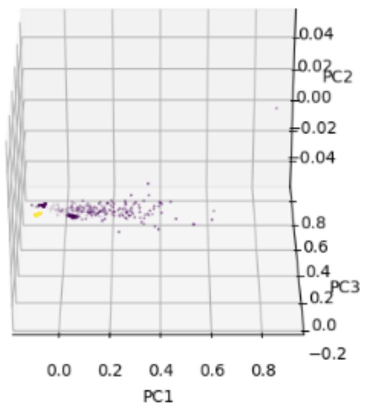
Table VII: PCA results for x86 and Arm Cortex-A72 performance counter data: principal components variance and features contribution

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



a.





b.

Fig. 6 : 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

C. Feature selection

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. 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 algorithms from the scikit library to determine feature importance. LASSO (Least Absolute Shrinkage and Selection Operator) , a statistical formula with the main purpose of feature selection and regularization of the data model. 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 VIII and Table IX.

TABLE VIII: 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 IX: 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

D. Performance Comparison

We integrated the performance counter events in Tables VIII and IX into our detection system and did a comparison of the SVM performance results.

Table X: SVM Performance Comparison On X86 - Negative Results

Features/ Negative results	PRECISION	RECALL	F1-SCORE	SUPPORT
6 features [3]	0.987	0.982	0.984	7381
35 features	1.000	1.000	1.000	7347
11 principal comp	1.000	1.000	1.000	7353
7 features selection	1.000	0.999	0.999	7291
11 features selection	1.000	1.000	1.000	7393

Table VIIII: SVM Performance Comparison On X86 - Positive Results

Features/ Negative results	PRECISION	RECALL	F1-SCORE	SUPPORT
6 features [3]	0.978	0.983	0.981	5922
35 features	1.000	1.000	1.000	5956
11 principal comp	1.000	1.000	1.000	5950
7 features selection	0.999	1.000	0.999	6012
11 features selection	1.000	1.000	1.000	5910

Table X shows SVM performance for negative results (normal applications) comparing the same sets of features discussed previously:

- Row "6 features" - the baseline manual selection used in Reference [3]
- Row "35 features" - all 35 relevant features used in detection, prior to feature selection
- Row "11 principal components" - the key principal components accounting for 90% variation
- Row "11 feature selection" – a reduced set of 11 features selected from the 35 features using filtering and wrapper approaches to feature selection, specifically Lasso and Gradient Boost feature selection methods from scikit
- Row "feature selection" - the top 7 features selected from the 35 features using filtering and wrapper approaches to feature selection, specifically Lasso and Gradient Boost feature selection methods from scikit

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

A comparison of results shows that the new list of features proposed in this paper, based on principal component analysis and feature selection algorithms, provides significantly better results than [3] and with similar performance to the study with the larger number of counters.

VI. 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. Our detection technique has been experimentally shown to be effective for Linux based operating systems in either an edge processing or cloud based environment communicating with one or more embedded system end nodes based on our prototype implementation and associated experimental results.

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 variant.

A more robust method for selecting event counters was investigated and validated against the original SPECTRE attack on x86 and Cortex-A72. 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.

ACKNOWLEDGMENT

We would like to thank Xida Ren and Professor Ashish Venkat from the University of Virginia for their collaboration on this research related to micro-op cache attack proof of concept.

REFERENCES

[1] Thomas M. Conte, Erik P. DeBenedictis, Avi Mendelson, and Dejan Milojičić, "Computers to avoid SPECTRE and meltdown," *IEEE* Computer , vol. 51, iss. 4, April 2018.

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

[3] 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.

[4] 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.

[5] A spectre proof of concept for web, Blog March, 2021 https://security.googleblog.com/2021/03/a-spectre-proof-of-concept-for-spectre.html

[6] 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,

[7] 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"

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

[9] 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