2018 30th International Symposium on Computer Architecture and High Performance Computing

Online Detection of Spectre Attacks Using Microarchitectural Traces from Performance Counters

Congmiao Li and Jean-Luc Gaudiot
Electrical Engineering and Computer Science
University of California, Irvine
Irvine, USA
congmial@uci.edu, gaudiot@uci.edu

***Abstract*— To improve processor performance, computer architects have adopted such acceleration techniques as speculative execution and caching. However, researchers have recently discovered that this approach implies inherent security flaws, as exploited by Meltdown and Spectre. Attacks targeting these vulnerabilities can leak protected data through side channels such as data cache timing by exploiting mis-speculated executions. The flaws can be catastrophic because they are fundamental and widespread and they affect many modern processors. Mitigating the effect of Meltdown is relatively straightforward in that it entails a software-based fix which has already been deployed by major OS vendors. However, to this day, there is no effective mitigation to Spectre. Fixing the problem may require a redesign of the architecture for conditional execution in future processors. In addition, a Spectre attack is hard to detect using traditional software-based antivirus techniques because it does not leave traces in traditional log files. In this paper, we proposed to monitor microarchitectural events such as cache misses, branch mispredictions from existing CPU performance counters to detect Spectre during attack runtime. Our detector was able to achieve 0% false negatives with less than 1% false positives using various machine learning classifiers with a reasonable performance overhead.**

Keywords— security, malware detection, microarchitectural features

I. INTRODUCTION

Information can be leaked to unprivileged parties through unintended side channels. It is important to protect information from unauthorized access to ensure security. In microarchitectural side-channel attacks, malicious processes attempt to interfere with the victim through shared microarchitectural resources. The interference pattern such as cache timing [1-4], branch prediction history [5,6], or Branch Target Buffers [7,8] can then be exploited to infer secrets.

Speculative execution improves performance by executing the predicted path prematurely before the actual path is known definitively. Spectre attacks [9, 10] exploit speculative execution by tricking the processor into taking the wrong branch, while instructions associated with the malicious branch execution path can be carefully crafted to leak the victim's memory or register content through microarchitectural side channels. Kocher *et al.* [10] demonstrated the use of cache side channel for their attack implementation. It should be noted that the KAISER software patch [11] to mitigate Meltdown attack cannot defend against Spectre attacks.

Acknowledgments -This work is partly supported by the National Science Foundation (NSF) under Grant No. CCF-1763793/3654. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

Traditional antivirus (AV) software scans suspicious instructions in binaries or traces in system logs files statically to detect malicious attacks. Unfortunately, Spectre usually does not leave traces in system log files. Thus, it is difficult to detect Spectre with a static analysis. However, recent research has shown that malicious software and attacks can be detected using dynamic microarchitectural execution patterns from widely available hardware performance counters (HPC) in modern processors [12]. This is done by adopting offline analysis based on various supervised machining learning algorithms to show that malware classification is possible using complete trace of the program behavior after execution.

In this paper, we propose to use microarchitectural features to detect Spectre while the attack is in progress. The hardware performance counter-based detection approach allows for the monitoring of the dynamic behavior of the system at the hardware level with low overhead and allows the system administrator to effectively catch Spectre in real time. Our methodology consists in collecting microarchitectural traces in a clean desktop environment and in a system under attack respectively, then use machine learning methods to train and test classifiers to detect the attack. Unlike previous offline detection method using the entire program traces after attack [12], we propose to use the Weighted Moving Average (WMA) of the time series data from the output of classifier to make successive decisions over sliding windows of program execution in real time. In our experiment, the online detector can catch 100% of the attacks with 0.77% false positives.

The rest of the paper is organized as the follows. We first explain how Spectre attacks work in section II. In section III, we introduce our proposed online detection approach. In section IV, detailed experimental setup is described. The detection performance results are presented in section V. Finally, we conclude this paper in section VI.

II. BACKGROUND

Meltdown and Spectre exploit critical modern processor design flaws to allow attackers to steal sensitive information from users' devices through microarchitectural side channels. Mitigation of Meltdown involves changes in kernel code to further isolate the kernel memory from user-mode processes. Such remedies have been released by OS vendors through software kernel patches.

978-1-5386-7769-8/18/$31.00 ©2018 IEEE
DOI 10.1109/SBAC-PAD.2018.00018

A. Spectre

However, there are still no efficient software patches for Spectre attacks until now. As reported in [9, 10], the attack has two variants: bounds check bypass and branch target injection. The first variant exploits conditional branch mispredictions and the second targets indirect jump target predictions. For example, the victim function in Listing 1 receives integer *x* from an untrusted source. The function does a bound check on *x* to prevent the process from reading unauthorized memory outside *array1* to ensure security. However, speculative execution can lead to out-of-bounds memory reads. Suppose the attacker makes several calls to *victim_function()* to train the branch predictor to expect taking the branch by feeding it with valid values of *x*, then calls the same function with an out-of-bound *x* that points to a target byte in the victim's protected memory.

```
void victim_function (size_t x) {
   if (x < array1_size) {
        temp &= array2[array1[x] * 512];
   }
}
```

Listing 1: Conditional Branch Example

A Spectre attack usually consists of three phases. It starts with the setup phase where the attacker prepares the side channel to leak the victim's sensitive information, and other necessary pre-requisites such as mis-training the branch predictor to take the erroneous execution path, *etc*. In the following phase, the attacker diverts confidential information from the victim's context to a microarchitectural side channel through speculative execution. Then during the final phase, the attacker gains access to the secret data through the side channel prepared in the previous stages.

B. Detection of Spectre

It is important to proactively detect a malicious attack and stop it at the earliest possible stage. For the example in Listing 1, the attacker calls the victim function multiple times that causes the condition to be true. Therefore, we speculate that the branch misprediction rate will be reduced during an attack. In addition, the secret data are leaked through cache side channel. For this, the attacker needs to flush the cache constantly, so the cache miss rate is likely to be increased. By monitoring the deviation of these two microarchitectural behaviors, it may be possible to detect an attack.

To further validate our hypothesis, we set up experimental attacks based on the proof of concept code in [10] and collected microarchitectural traces from HPC to analyze the results, as shown in the subsequent sections.

III. Proposed Online Detection Approach

In our proposed detection approach, we first collect microarchitectural features from HPC every *t* second, where *t* is the sampling period. For the variant of Spectre attack discussed in the previous section using a conditional branch, we choose to monitor 4 events including cache references, cache misses, branch instructions retired and branch mispredictions. The data are respectively collected in a clean environment and in an environment under Spectre attack. The data are then labeled to train the machine learning classifier to classify the input data for each sampling period *t.* At runtime, the output classification time series is fed into the online detection mechanism to decide whether the system is under attack. This section describes in detail the machine learning algorithms we used to train the classifier and our proposed online detection approach.

A. Machine Learning Classifiers

Machine learning (ML) can be used to train classifiers that determine the class to which a given data set belongs. We use supervised learning [13] to train the detector with a set of pre-labeled samples. We choose machine learning algorithms such as Logistic Regression (LR), Support Vector Machine (SVM), and Artificial Neural Networks (ANN) (also known as Multilayer Perceptron (MLP)), to build classifiers of increasing complexity. We collect data in 10 independent runs and use the same number (1200) of samples from both classes to avoid any bias. Then we randomly divide the collected data into training (80%) and test (20%) data, then separate training data into training (80%) and validation (20%) data.

B. Online Attack Detection

We propose to use online classification method to detect malicious behavior during attacks. We periodically collect microarchitectural features. The multidimensional data are then feed to a machine learning classifier to make consecutive decisions as to maliciousness. The problem of detecting malicious attacks in real time is essentially to make decisions according to the time series generated from the base classifier.

To smooth the fluctuated time series data, a Weighted Moving Average (WMA) is used to filter out noise for better decision making by assigning a weight factor to each element in the time series. Then we segment the data using a sliding window [14] to calculate the average of consecutive decisions within the current window. If the average is above a certain threshold, we consider it a malicious attack. For our experiments, we chose the window size to be 10 and the sampling period to be 0.1 second, which means our detector makes a decision every second. Note that these parameters should be tuned for different hardware systems. In general, a larger window size and smaller sampling period may give better detection accuracy but a larger performance overhead. For our system, we choose the above numbers so as to reach a satisfiable detection accuracy without incurring a major slowdown of the system.

C. Evaluation of Detection Performance

A key criterion to evaluate the detection performance is the accuracy of the model used to make decisions on previously unseen data. To characterize accuracy, we use metrics such as False Positives (FP) which is the percentage of misclassified malicious instances, and False Negatives (FN) which is the percentage of misclassified normal instances. A detection approach with good performance is expected to minimize both.

To visualize the tradeoff between percentage of correctly identified malicious instances and the percentage of normal instances misclassified, we use Receiver Operating

Characteristic (ROC) graphs that plot True Positives (TP = 100% - FN) against FP. In addition, to compare the performance of different models, we could compute and compare the area under the ROC curve for each model. The Area Under Curve (AUC) score (also known as c-index) provides a quantitative measurement of how well an attack detection approach performs (higher AUC value means better performance).

IV. Experimental Setup

In this section, we describe the details of the microarchitectural data collection mechanisms, as well as the system settings under attack and in normal conditions.

A. Data Collection Mechanism

We run the attack on a typical personal laptop with Debian Linux 4.8.5 OS on Intel® Core™ i3-3217U 1.8 GHz processor with 3MB cache and 4GB of memory. The Intel processor contains a model-specific Performance Counter Monitor (PCM) and can be configured to count up to four different hardware events at the same time. According to the discussion on the nature of Spectre attacks in Section II, we choose 4 available events for our system, which are Last-level cache reference events (LLC references), Last-level cache misses events (LLC misses), Branch instruction retired events (branches), and Branch mispredict retired events (branch mispredictions). We use the standard profiling infrastructure on Linux, *perf* tools, to obtain system-wide performance counter data.

B. Test Environment Setup

In the clean environment, we sought to create realistic scenarios by randomly browsing popular websites with FireFox in different orders, and streaming videos from browser plug-ins. In addition, we also ran a text editor to read and edit files. For data collection when the system is under malicious attack, we launch the Spectre proof of concept attack on top of the normal running applications. System status is reset after each run to ensure the measurements are independent across different clean and exploit runs. We collect overall performance counter data across the system rather than individual process to make the classification problem more difficult and closer to real world setup.

V. Results

In the experiment described in section III, we periodically collect data from the four performance counters at the same time. In this section, we first analyze the collected raw data to see if it is feasible to differentiate measurements in clean environment and those under attack by visualizing the distribution of data. Then we use different machine learning algorithms to train the classifier and build the real-time attack detector using sliding window approach discussed previously.

A. Data Distribution Analysis

Our data collection mechanism produces 4-dimensional time series data. Each sample contains event counts for branch mispredictions, LLC misses, branches, LLC references during the sampling period. We also calculate the branch miss rate (1) and the LLC miss rate (2) for each interval as defined below:

$$branch\ miss\ rate = branch\ mispredictions / branches \quad (1)$$

$$LLC\ miss\ rate = LLC\ misses / LLC\ references \quad (2)$$

We analyze the feasibility to distinguish the data collected using more than one feature by plotting the sample points in 3D graphs with each dimension corresponding to one feature. Fig. 1 shows the distribution of normal and malicious sample points using LLC references, LLC misses and branch miss rate. We can see the data points of two different classes distribute in two different regions and the boundary between the two is obvious. Therefore, we believe it is possible to use the chosen microarchitectural features to detect Spectre attacks.

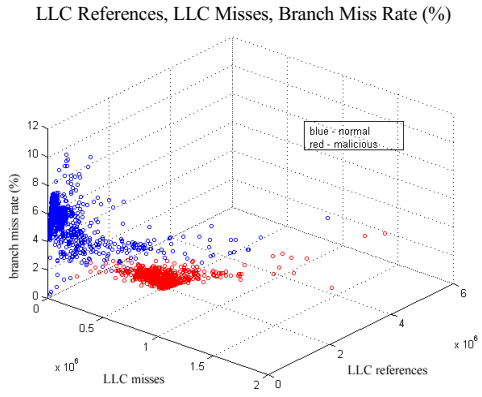

Fig. 1. Distribution of LLC references, LLC misses and branch miss rate features.

B. Online Detection Performance

We use 3 different machine learning algorithms to train the base classifier, then smooth the output time series with WMA, and finally build the online detector based on a sliding window approach.

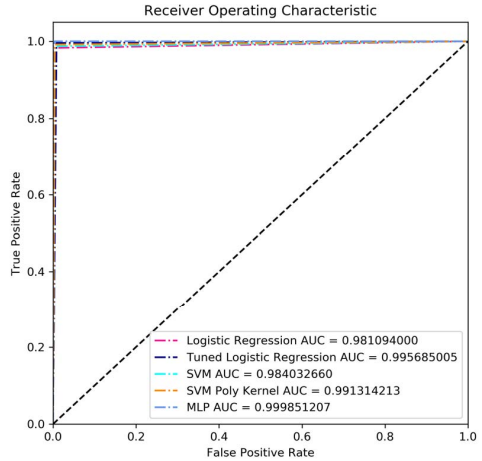

Fig. 2. ROC for online detectors using different classifiers

A simple model is first built with default parameters for each classifier. To further enhance the detection accuracy, different parameters are tuned for each ML model. We use a randomized search over different parameters to find the best combination, where each setting is sampled over a distribution over possible parameter values. Compared with an exhaustive search, it is less computationally expensive and gives a result close to optimal.

To evaluate the performance of online detection based on different classifiers, ROC curves are plotted. Fig. 2 shows the ROC curves for all the trained classifiers in our experiment. As we can see in the graph, there is a trade-off between false positives and true positives; the detector is able to catch more malicious attacks if we allow higher false positives. To choose the best configuration, we pick the point on the curve at the top left corner which gives the lowest combined number of false negatives and false positives. We can see that they all perform very well with an AUC above 0.9. Overall, MLP outperforms other classifiers with the highest AUC value. This is as expected because MLP is more flexible than other methods used. However, it requires a longer training time.

TABLE I. PERFORMANCE FOR DIFFERENT CLASSIFIERS

Classifier	AUC	FP (%)	FN (%)	Training Time (sec)
LR	0.9810939999	3.83	3.40	0.04
Tuned LR	0.9956850054	1.15	2.43	0.04
SVM	0.9840326600	0.77	2.43	0.06
SVM with Polynomial Kernel	0.9913142134	0.77	0.97	9.8
MLP	0.9998512071	0.77	0	95

We compare the performance of each classifier quantitatively using the AUC index and choose the best point on the ROC which gives the minimum FP and FN as shown in Table I. For the best case using MLP, the detector is able to achieve 0% false negatives with only 0.77% false positives.

VI. CONCLUSIONS

This paper proposes to detect Spectre attacks by monitoring microarchitectural features deviations as the attack exploits vulnerabilities in modern processor hardware design such as speculative execution and cache side channels. The features are collected from hardware performance counters widely available in processors. An online detection method is adopted to detect malicious behaviors at an early stage of the attack rather than offline detection after the damage has been done. The experimental results show a promising detection accuracy with only 0.77% of false positives with 0% false negatives using a trained multilayer perceptron classifier. As complete mitigation to the Spectre is challenging, it is more practical now to detect such attacks proactively.

There are many variants of Spectre according to different types of hardware design flaws and side channels being exploited. New variants are discovered constantly and recently researchers have discovered a new speculative store bypass vulnerability [15]. However, all the different variants use a side channel to infer confidential information in the final stage of the attack. In addition, stealth mode Spectre attacks are usually ineffective. We thus believe it is possible to detect malicious behaviors by monitoring changes in these hardware side channels. This research shows the feasibility to detect Spectre using such approach. Future research can be carried out for other variants of the attack under different configurations such as in servers and virtual machines environments.

REFERENCES

[1] D. A. Osvik, A. Shamir and E. Tromer, "Cache attacks and Countermeasures: the Case of AES," Cryptology ePrint Archive, Report 2005/271, 2005.

[2] D. J. Bernstein, "Cache-timing Attacks on AES", http://cr.yp.to/antiforgery/cachetiming20050414.pdf, 2005.

[3] Y. Yarom and K. Falkner, "FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack," USENIX Security, San Diego, CA, US, Aug 2014, pp.719–732.

[4] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," IEEE Symposium on Security and Privacy (S&P), May 2015, pp. 605–622.

[5] Aciiçmez, O., Gueron, S., and Seifert, J.-P., "New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures," 11th IMA International Conference on Cryptography and Coding (Dec. 2007), S. D. Galbraith, Ed., vol. 4887 of Lecture Notes in Computer Science, Springer, Heidelberg, pp. 185–203.

[6] Aciiçmez, O., Koç, Çetin Kaya., and Seifert, J.-P., "Predicting secret keys via branch prediction," Topics in Cryptology CT-RSA 2007 (Feb. 2007), M. Abe, Ed., vol. 4377 of Lecture Notes in Computer Science, Springer, Heidelberg, pp. 225–242.

[7] Evtyushkin, D., Ponomarev, D. V., and Abughazaleh, N. B., "Jump over ASLR: attacking branch predictors to bypass ASLR," MICRO (2016), IEEE Computer Society, pp. 1–13.

[8] Lee, S., Shih, M., Gera, P., Kim, T., Kim, H., and Peinado, M., "Inferring fine-grained control flow inside SGX enclaves with branch shadowing," 26th USENIX Security Symposium, USENIX Security 2017, pp. 557–574.

[9] Horn, J., "Reading privileged memory with a side-channel," https://googleprojectzero.blogspot.com/2018/01/reading-privileged-memory-with-side.html, 2018.

[10] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y., "Spectre attacks: Exploiting speculative execution," ArXiv e-prints, Jan. 2018.

[11] Gruss, D., Lipp, M., Schwarz, M., Fellner, R., Maurice, C., and Mangard, S., "KASLR is dead: long live KASLR," International Symposium on Engineering Secure Software and Systems, Springer, pp. 161–176, 2017.

[12] Demme, J., Maycock, M., Schmitz, J., Tang, A., Waksman, A., Sethumadhavan, S., Stolfo, S., "On the feasibility of online malware detection with performance counters," Proceedings of the International Symposium on Computer Architecture (ISCA), 2013.

[13] J. Frank, "Machine learning and intrusion detection: Current and future directions," in Proc. National 17th Computer Security Conference, Washington,D.C., October 1994.

[14] Vafaeipour, M., Rahbari, O., Rosen, M.A., Fazelpour, F. & Ansarirad, P., "Application of sliding window technique for prediction of wind velocity time series," International journal of Energy and environmental engeering (springer), 5,105-111, 2014.

[15] Horn, J., "Speculative execution, variant 4: speculative store bypass", https://bugs.chromium.org/p/project-zero/issues/detail?id=1528, May 2018.