Performance Statistics and Learning based Detection of Exploitative Speculative Attacks

Swastika Dutta*
Sayan Sinha*
swastika@iitkgp.ac.in
sayan.sinha@iitkgp.ac.in
Indian Institute of Technology
Kharagpur, India

ABSTRACT

Most of the modern processors perform out-of-order speculative executions to maximise system performance. Spectre and Meltdown exploit these optimisations and execute certain instructions leading to leakage of confidential information of the victim. All the variants of this class of attacks necessarily exploit branch prediction or speculative execution. Using this insight, we develop a two step strategy to effectively detect these attacks using performance counter statistics, correlation coefficient model, deep neural network and fast Fourier transform. Our approach is expected to provide reliable, fast and highly accurate results with no perceivable loss in system performance or system overhead.

CCS CONCEPTS

• **Security and privacy → Security in hardware**; *Hardware attacks and countermeasures*; Side-channel analysis and countermeasures; • **Computing methodologies → Machine learning**; *Machine learning approaches*; Neural networks.

KEYWORDS

Spectre, Meltdown, LSTM, Hardware Performance Counters, Correlation Coefficient, Fourier Transform

ACM Reference Format:
Swastika Dutta and Sayan Sinha. 2019. Performance Statistics and Learning based Detection of Exploitative Speculative Attacks. In *Proceedings of the 16th conference on Computing Frontiers (CF '19), April 30-May 2, 2019, Alghero, Italy.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3310273.3322832

*Both authors contributed equally to this research.

CF '19, April 30-May 2, 2019, Alghero, Italy

ACM ISBN 978-1-4503-6685-4/19/05…$15.00
https://doi.org/10.1145/3310273.3322832

1 INTRODUCTION

Modern processors use various optimisation techniques in order to improve system performance. Several design techniques have facilitated the increase in processor speed over the past decades. One such advancement is speculative execution, which makes use of branch prediction and helps in effective utilisation of time and resources.

Though manufacturers take measures to avoid security defects due to such optimisations, such efforts prove to be vulnerable at times. Recently, the vulnerabilities of Spectre [8] and Meltdown [9] have been exposed in modern processors. It makes use of observable side effects beyond the computation's nominal outputs to read data from areas of memory outside the scope of user access. It is unknown if these vulnerabilities have been put into practice to retrieve secret information since they do not leave any trace post-execution. Research in this area has paced up considerably since the methodology of Spectre and Meltdown is now known to the entire world and is prone to be exploited. We propose a practical detection strategy with no necessary maintenance at the user end. It does not require any kernel level modification and leaves scope for CPU optimisations to take place and maintain healthy system performance in a varied range of environments.

We propose using Hardware Performance Counters (HPCs) to extract the features of a system during the occurrence of speculative execution. Our detection strategy involves the use of a long short term memory (LSTM) model, Savitzky Golay filter [10], Input / Output Control and fast Fourier transform analysis to record and analyse hardware performance statistics of the system and thereby successfully detect these attacks.

2 RELATED WORKS

HPCs have been used extensively in the past to detect malicious activities. For instance, Chiappetta et al. [7] proposed

using HPCs to detect side channel attacks. HPCs to detect malicious modification of firmware in embedded control systems and Signature-based ROP Detection to detect Return Oriented Programming (ROP) attacks have also been popular. However, these works have mainly focused on exposing the vulnerability of the system, rather than providing a solution to guard against speculative attacks in real time.

Detection of cache-based side channel attack has been long studied. Machine learning based approaches on hardware performance counters have also been discussed in previous research [6]. Most other approaches lack experimentation on deep learning methods and do not focus on time series data.

Previous attempts have been made to safeguard systems against Spectre and Meltdown vulnerabilities. These are mostly kernel level patches which restrict specific optimisation steps. It has been shown such patches (such as KPTI [3] in Linux for Meltdown) can have up to 800% overhead in CPUs without hardware pcid support [4]. Recent efforts have been made to detect the exploitation of these specific vulnerabilities using hardware performance counters, such as the blog post by Check Point Research [2] or TrendMicro [1]. However, these efforts mainly suggest measures to tackle this problem, rather than being a full-fledged practical implementation which can be used in large scale. In this paper, we introduce an automated and robust detection strategy which is divided into two stages in order to reduce overhead and make the approach more objective.

3 OUR METHOD

We shall exploit the fact that a repetitive prime probe process is essential for reading cache memory post speculative execution to develop a detection strategy. A repetitive training-and-fooling process makes it sufficient to conclude the presence of the Spectre vulnerability, while its absence suggests the presence of Meltdown.

By and large, our detection strategy proceeds into the following phases, as shown in Figure 5. We present here a detailed methodology for the detection of these anomalies using the above strategy.

SET UP PHASE

1. Implementation of benign programs

To represent a prototype of a system being exploited by such vulnerabilities, we design some programs that mimic the activities of exploitative speculative attacks.

Namely, we implement the prime probe and training-and-fooling processes because they are necessary for the respective exploitative attacks.

In the benign prime probe process, there is an attempt to access the entire range of possible ASCII characters till the character present in the cache (and subsequently, the whole string of characters or data) is discovered. Analogous to this, we design a program in which we allocate a matrix with some values and flush its contents. We then try to access them repeatedly, leading to cache misses. A pseudo code of the implementation is presented in Algorithm 1.

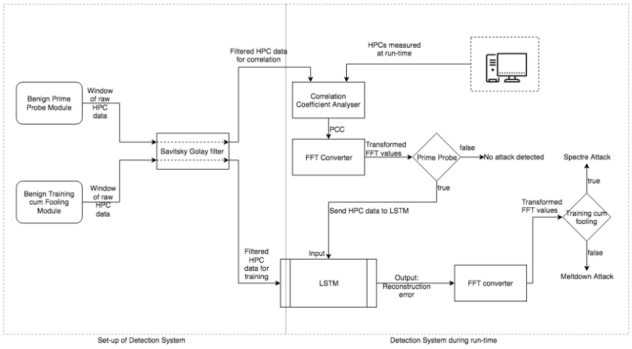

Figure 5: Flow of control in our detection strategy

Algorithm 1 Benign prime probe

```
1: Input: num_iter, ArrayA[matrix_size]
2: count ← 1
3: while count ≤ num_iter do
4:     count ← count + 1
5:     Flush cache from A[0] to A[matrix_size − 1]
6:     counter ← 0
7:     while counter ≤ matrix_size − 1 do
8:         temp ← A[counter]
9:         counter ← counter + 1
10:    end while
11: end while
```

Similarly, we design benign training-and-fooling module where the decision taken by an if statement is predicted to be true for certain number of times. This trains the branch predictor to predict true for successive instructions. We then pass an argument which should be false. Along with this, we keep a statement body inside the if block, which would access illegal memory only if the decision taken is negative. We again collect hardware performance statistics for this event. Pseudo code is presented in Algorithm 2.

2a. Observing the system behaviour using IOCTLs and pre-defined modules

The mentioned vulnerabilities mostly work on the basis of the hardware events of cache miss and branch miss. Hence we chose these performance statistics to begin with. These hardware counters do not introduce additional overheads and typically consume minimal resources since they are already built into the processors. The counters are incremented on an instruction-by-instruction basis, thus ensuring accurate results. The HPCs are observed using the popular tool, Input / Output Control (IOCTL) calls, available in Unix version 7+ and most Unix-like systems.

Algorithm 2 Benign training fooling

```
Input: num_iter_outer, num_iter_inner. ArrayA[array_size], const_1, const_2
count ← 1
while count ≤ num_iter_outer do
    Flush cache from A[0] to A[array_size − 1]
    count ← count + 1
    counter ← 1
    while counter ≤ num_iter_inner do
        if counter = num_iter_inner then
            index = array_size + const_1          ▷ index ≥ array_size
        else
            index = const_2                       ▷ index ≤ array_size − 1
        end if
        if index ≤ array_size − 1 then
            temp = A[index]
        end if
    end while
end while
```

Generally, there should be a decrease in branch misses as the system goes through the training process of the Spectre vulnerability. This is followed by a noticeable increase in cache-misses as the prime probe event takes place, as observed in Figure 1.

The optimal sampling period obtained by validation is 7,500 instruction counts and thus, we measure HPCs with very high frequency. This makes our data susceptible to arbitrary changes in values at some given instances of time. Hence, we shall filter the data collected in order to eliminate such errors and system noises present in the data.

If the value of sampling period is low (around 1000), we notice remarkable system noise, as evident from Figure 6. The HPCs obtained at 7500 instruction counts is shown in Figure 7 for comparison. Also, if the count is kept significantly high, we obtain a very low number of samples (in the order of ten thousand) which is not sufficient to train the LSTM model.

2b. Pre-processing acquired data using Savitzky-Golay filter

Savitzky-Golay (Savgol) filter [10] is a digital filter that uses the method of linear least squares. We chose Savgol filter since it has less overhead and works faster than other popular filters (like the Kalman filter), which is essential for real time measurements. We fit the hardware statistics data into a polynomial of degree seven, which was experimentally determined.

3. Calculation of the correlation coefficient between our prime probe module and speculative attacks

In our strategy, we create a sliding window of 3,000 data points of run time HPCs and slide it across the dummy cache miss values obtained in Step 1. The correlation detection model first takes an input sequence (X) from i to $i + 3,000$ time instants of cache miss values from the system and generates the Person correlation coefficient (PCC) corresponding to that sliding window.

4. Learning Time-Series data using LSTM

The HPC data obtained is a time series data and is expected to have a specific temporal pattern as shown in the above Figure 2. Using the LSTM network, we try to input a window of data points and predict the next expected value of a data point.

Model description

Objective: To predict the 50^{th} data point from 49 previous data points of cache misses and branch misses.
Input X**:** HPC data (branch-misses and cache misses) collected during execution of pre-defined modules using 7,500 instruction sampling count, divided into sliding windows of length 50. The input feature vector is two-dimensional owing to the presence of both branch miss and cache miss values.

Training: Without loss of generality, we had chosen 50 trace points for our experiments. We then shift the window by one-time interval repeatedly to consider the next consecutive 50 sample points for learning. Once the training is completed, for an anomalous sequence, the LSTM attempts to reconstruct the 50^{th} data point from given 49 data points.

Table 1: Architecture of our LSTM Model:

Layer number	Layer Type	Input Shape	Output Shape
1	LSTM	(None,49,2)	(None,49,32)
2	LSTM	(None,49,32)	(None,49,16)
3	LSTM	(None,49,16)	(None,1,2)

Inference: The model designed is expected to predict 50^{th} value depending on 49 previous samples, if the system is under execution of a training-and-fooling module.

DETECTION PHASE:

1. Detection of prime probe using correlation coefficient analyser

We shall find the PCC value between the dummy prime probe module and speculative attack modules using the model designed in the setup phase. A prime probe process may be under execution whenever the PCC value obtained is above a particular threshold. Since the prime probe event keeps happening periodically, we expect a plot as shown in Figure 1. In Spectre attacks, prime probe occurs after training-and-fooling modules and hence, we expect system HPCs to have high PCC values at such time instances, as shown in Figure 8 (the red lines mark the training-and-fooling periods).

2. FFT Analysis on correlation coefficient:

We use fast Fourier transform (FFT) which helps us analyse if hardware performance statistics repeat at regular intervals, thus confirming the presence of repetitive system processes. We expect the correlation coefficient obtained in respective windows to be periodic unless it is a false alarm. We then proceed to the next step to determine the type of attack.

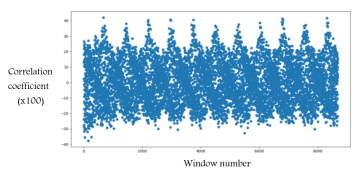

Figure 1

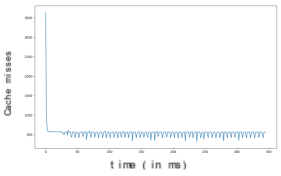

Figure 2

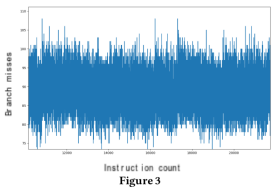

Figure 3

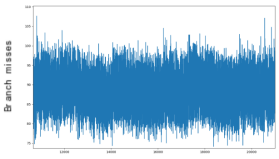

Figure 4

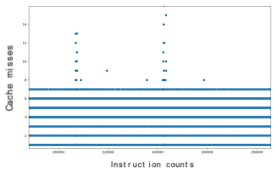

Figure 6

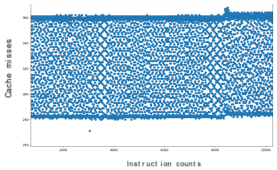

Figure 7

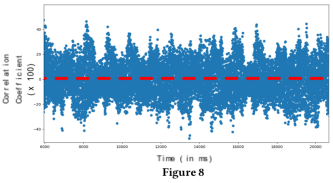

Figure 8

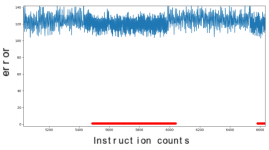

Figure 9

Figure (1) Plot of Pearson Correlation Coefficient in a window vs window number during execution of Meltdown; (2) Plot of cache misses vs data points during execution of spectre; (3) Plot of branch misses vs datapoint number during execution of spectre branch misses before smoothing of data using Savistzky Golay filter; (4) Plot of branch misses vs datapoint during execution of Spectre after smoothing of data using Savgol filter; (6) Plot of cache misses vs datapoint during execution of Meltdown at 1000 instruction sampling rate; (7) Plot of cache misses vs datapoint during Meltdown at 7500 sampling rate; (8) Plot of correlation coefficient vs time during execution of Spectre. The red lines mark the previously noted training-and-fooling period; (9) Plot of reconstruction error obtained from LSTM while execution of Spectre versus datapoint. The red lines mark previously noted training-and-fooling periods.

3. FFT analysis on reconstruction error:

Using the LSTM model trained in the setup phase, we find the reconstruction error of HPCs obtained in real time. The reconstruction error is the L2 norm between the true value of 50^{th} and the value predicted by our model. The reconstruction error should be substantially low at the points where any training-and-fooling processes are encountered, as shown in Figure 9. These reconstruction errors are then analysed by FFT analysis. training-and-fooling processes in Spectre attacks are periodic, and hence, we expect the FFT Analysis to have maxima at a particular frequency. We can thereby conclude the presence of training-and-fooling modules, or a Spectre attack, in the system.

4 EXPERIMENTATION AND RESULT

We first train our LSTM with the HPCs obtained by our designed benign training-and-fooling process with IOCTL calls. After this, we run samples of Spectre and Meltdown programs and attempt to detect and differentiate between them. We perform the experiments on a sandbox environment having the specification 64-bit Linux 3.16.0-77-generic, with 7.7 GiB of memory and Intel Core i5, 1.6 GHz and 8 CPUs. Our experimentation is discussed in further details as follows:

Detection of prime probe

The hardware performance statistics of the system are recorded dynamically and restructured into samples of 3,000-time instants and fed into the PCC model. If the correlation coefficient calculated is greater than a threshold, we feed the PCC values into the FFT analyser. This threshold is determined according to the 3σ rule of thumb which is used in other similar experiments [5]. The value of $+3\sigma$ obtained in our research was 0.297. This value gave us a precision of 0.786 and recall of 0.932 for detection of prime probe event. We shall confirm that a prime probe module is under execution if two distinct peaks are observed, as in Figure 10 and Figure 11. Experimentally, we noticed that we could conclude the presence of a prominent peak if the ratio between the highest peak and the second highest peaks in the neighbourhood is more than 1/4. In these figures, the frequency of one of the observed peaks is due to the ongoing prime probe process, in respective modules (17962.3 instruction counts for Spectre and 1862.3 instruction counts for Meltdown attacks). However, the other average frequency with a peak in both the plots is observed to be nearly the same, i.e., 7396.9 instruction counts and 7439.4 instruction counts for Spectre and Meltdown attacks respectively. This common peak is due to the usage of same sampling period in IOCTL calls, i.e. 7500 instruction counts for both Spectre and Meltdown. The instruction sampling results in cache events which is reflected in the FFT analysis. Upon confirmation of the presence of exploitative speculative attacks, we move to our next step of experimentation to differentiate between the various kinds of such attacks.

Detection of training-and-fooling process

The reconstruction error of the LSTM model discussed under Section 5 is plotted as in Figure 9. As expected, the reconstruction error has been observed to decrease during the

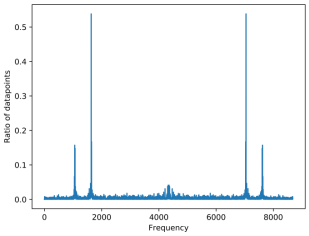

Figure 10

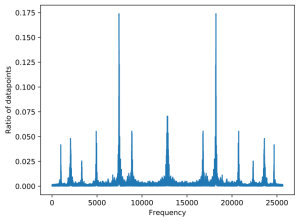

Figure 11

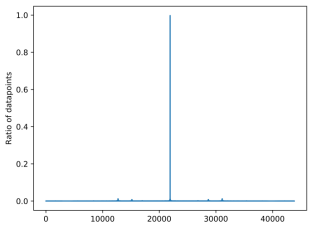

Figure 12:

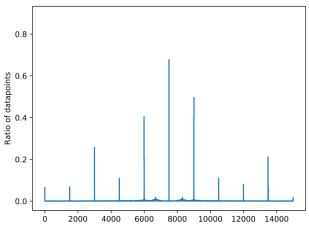

Figure 13

Figure (10) Plot of ratio of data points vs frequency obtained by FFT analysis on correlation coefficient data during execution of Meltdown; (11) Plot of ratio of datapoints versus frequency obtained by FFT analysis on correlation coefficient data during execution of Spectre; (12) Plot of ratio of datapoints vs frequency obtained by FFT analysis on reconstruction error in LSTM during execution of Spectre; (13) Plot of ratio of datapoints vs frequency obtained by FFT analysis on reconstruction error in LSTM during execution of Meltdown.

ongoing training-and-fooling process. Experimentally, we observe the average value of reconstruction error during execution of training-and-fooling modules is 119.232, while it significantly increases to an average value of 136.011 otherwise. We shall use FFT analyser to determine the repetitive nature of the reconstruction error values. HPCs recorded during Spectre attack seemed to have a prominent peak (21542.7 instruction counts) as shown in Figure 12, confirming the presence of a training-and-fooling process. The peaks are obtained by simple thresholding as mentioned earlier.

In Figure 13, the FFT analysis lacks a prominent peak due to the absence of a training-and-fooling process. If the system is subjected to training-and-fooling processes along with prime probe, we can conclude that the system had been under a Spectre attack, while the presence of prime probe alone shall indicate Meltdown attack. The results of the FFT analysis have been presented in Table II.

Table 2: Observation on no of peaks:

Observation type	Expected no of high peaks	Observed no of high peaks
Meltdown correlation	256	250
Spectre correlation	1000	1040
Spectre reconstruction	5000	5096
FFT on Meltdown correlation	2	2
FFT on Spectre correlation	2	2
FFT on Spectre LSTM reconstruction	1	1

5 CONCLUSION

In this paper, we explored the effects of execution of exploitative speculative attacks such as Spectre and Meltdown on HPCs in normal systems and used them to detect potential threats. We take a two-level detection framework, the first one being trained to detect prime probe modules, which is necessary for all speculative attacks and is based on a PCC model. As ascertained by the results, our models are capable of detecting any exploitative speculative execution attack during run-time, without any noticeable performance overhead.

Acknowledgement. We thank Debdeep Mukhopadhyay (debdeep@iitkgp.ac.in) and Manaar Alam (alam.manaar@iitkgp.ac.in) for their continued support and guidance.

REFERENCES

[1] [n. d.]. Detecting Attacks that Exploit Meltdown and Spectre with Performance Counters - TrendLabs Security Intelligence Blog. https://blog.trendmicro.com/trendlabs-security-intelligence/detecting-attacks-that-exploit-meltdown-and-spectre-with-performance-counters/.

[2] [n. d.]. Detection of the Meltdown and Spectre Vulnerabilities - Check Point Research. https://research.checkpoint.com/detection-meltdown-spectre-vulnerabilities-using-checkpoint-cpu-level-technology/.

[3] [n. d.]. KPTI - the new kernel feature to mitigate "meltdown" - Fedora Magazine. https://fedoramagazine.org/kpti-new-kernel-feature-mitigate-meltdown/.

[4] [n. d.]. Linux Meltdown patch: 'Up to 800 percent CPU overhead', Netflix tests show | ZDNet. https://www.zdnet.com/article/linux-meltdown-patch-up-to-800-percent-cpu-overhead-netflix-tests-show/.

[5] Manaar Alam, Sarani Bhattacharya, Swastika Dutta, Sayan Sinha, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. 2019. RATAFIA: Ransomware Analysis using Time And Frequency Informed Autoencoders. In *IEEE International Symposium on Hardware Oriented Security and Trust (2019)*. (in press).

[6] Manaar Alam, Sarani Bhattacharya, Debdeep Mukhopadhyay, and Sourangshu Bhattacharya. 2017. Performance Counters to Rescue: A Machine Learning based safeguard against Micro-architectural Side-Channel-Attacks. (2017).

[7] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing* 49 (2016), 1162–1174.

[8] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).

[9] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. 2018. Meltdown: Reading kernel memory from user space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 973–990.

[10] Ronald W Schafer. 2011. What is a Savitzky-Golay filter?[lecture notes]. *IEEE Signal processing magazine* 28, 4 (2011), 111–117.