Challenges in Detecting an "Evasive Spectre"

Congmiao Li, *Student Member, IEEE*
and Jean-Luc Gaudiot, *Life Fellow, IEEE*

Abstract—Spectre attacks exploit serious vulnerabilities in modern CPU design to extract sensitive data through side channels. Completely fixing the problem would require a redesign of the architecture for conditional execution which cannot be backported. Researchers have proposed to detect Spectre with promising accuracy by monitoring deviations in microarchitectural events using existing hardware performance counters. However, the attacker may attempt to evade detection by reshaping the microarchitectural profile of Spectre so as to mimic benign programs. This letter thus identifies the challenges in detecting "Evasive Spectre" attacks by showing that the detection accuracy drops significantly after the attacker inserted carefully chosen instructions in the middle of an attack or periodically put the attack to sleep at a frequency higher than the victim's sampling rate when operating the attack at a lower bandwidth, yet with reasonable success rate.

Index Terms—Evasive malware, microarchitectural attacks, security

1 Introduction

Modern processors use speculative execution to improve performance by pre-executing a predicted path before knowing the actual condition. However, this can open some unintended microarchitectural side channels. Recent Spectre attacks [1] exploit speculative execution to leak confidential information through unintended cache side channels by tricking the processor into taking a carefully crafted malicious branch.

Current software patches usually incur significant performance overhead and hardware fixes are not backportable to old machines. In addition, mitigation techniques may not prevent zero-day attacks. Therefore, it is important to proactively detect malicious attacks as an extra layer of defense. Recent research [2], [3], [4] has shown that malware can be detected by using dynamic microarchitectural execution patterns gleaned from existing Hardware Performance Counters (HPC). An HPC-based detector can effectively detect Spectre with high accuracy using machine learning classifiers [5], [6].

In this paper, we study how the original Spectre [1] could be even more maliciously updated to operate effectively without being detected by HPC-based classifiers. Previous research [7], [8], [9], [10] suggested generating *evasive malware* (also known as *mimicry attack*) by instruction insertion, code obfuscation, or calling of benign functions in between malignant payloads. Researchers [10] add instructions in the control flow graph of the malware in a way that does not affect the execution state of the program to evade HPC-based detectors. However, the inserted instructions may change the microarchitectural state such as the cache content of the victim. Compared with the above-mentioned evasive malware, developing evasive microarchitectural side channel attacks such as Spectre has additional requirements, because they are time sensitive and the attacker must ensure relevant microarchitectural status is unchanged in order to perform successful attacks. This research studies the feasibility of constructing evasive Spectre that is able to bypass HPC-based detector while maintaining a reasonable attack success rate, and also the trade-off between attack success rate and attack evasiveness. To achieve reasonable attack success rates, the attacker has to insert instructions or put the attack to sleep at coarser granularity than a basic block in the control flow graph. Therefore, We define atomic tasks (detailed definition in Section 3.2) and reshape the microarchitectural features in the granularity of atomic task level. If an atomic task is interrupted, the attack success rate would be greatly reduced. We also quantitatively compare different strategies to determine the best way an attacker could use to evade detection while maintaining a reasonable success rate and speed.

• *The authors are with the Department of Electrical Engineering and Computer Science, University of California Irvine, Irvine, CA 92697. E-mail: {congmial, gaudiot}@uci.edu.*

Manuscript received 23 Jan. 2020; accepted 12 Feb. 2020. Date of publication 20 Feb. 2020; date of current version 3 Apr. 2020.
(Corresponding author: Congmiao Li.)
Digital Object Identifier no. 10.1109/LCA.2020.2976069

2 Threat Model

We assume that the victim's machine is running an HPC-based malware detector such as proposed in [5] to defend from Spectre attacks. The detector monitors four microarchitectural features including Last-Level Cache references (LLC references), Last-Level Cache misses (LLC misses), branch instructions retired (branches) and branch mispredict retired (branch mispredictions) at a fixed sampling rate on a separated core. In future research, the condition will be relaxed for mixed sampling rate and the detector can be implemented in dedicated hardware to reduce performance overhead. We assume the attacker's goal is to reveal some confidential memory content on the victim machine without being detected as malware. To achieve this goal, we further assume the attacker can observe the behavior of the malware classifier from a machine with a similar HPC-based detector as the victim machine. The attacker can evade detection by changing the microarchitectural characteristics of the updated Spectre so as to behave like benign programs.

We assume the attacker knows the features being monitored by the malware classifier. This is reasonable because the attacker knows that the original Spectre [1] would cause increased cache misses and reduced branch mispredictions. However, the attacker does not know the sampling period of the detector. Yet this can be reverse engineered (see [10]).

As a Spectre attack runs in a loop, we assume the attacker can slow down the attack by calling the victim function/API at specific intervals. In addition, we assume the attacker can manipulate the performance counters by inserting instructions that reduce LLC cache misses and increase branch mispredictions by exploiting other vulnerabilities such as just-in-time code reuse attacks. This gives the attacker more privileges and could help us test the detector's resilience to evasion in extreme conditions.

Previous work [10] shows that the accuracy of HPC-based detectors decreases significantly as the number of instructions inserted in the original attack increases and assumes the attacker is interested in maintaining the performance of the attack. However, this did not consider how the inserted instructions may affect the success rate of the detector in inferring the correct content. Since Spectre is time sensitive, the inserted instructions may change or provide opportunities for other running programs to change the cache status and cause the attack to read the wrong content. Therefore, we further assume the attacker aims at maintaining a reasonable success rate.

3 Designing an "Evasive Spectre"

To avoid detection by HPC-based detectors, the attacker would seek to shape its microarchitectural trace to mimic that of benign programs. In so doing, the attacker would have to sacrifice the efficiency of the attack and perform it more slowly (reduced side channel bandwidth). However, slowing down Spectre could result in a failure of the attack. In this section, we first introduce the experimental

TABLE 1
Attack Success Rate After Interruption at Different Levels

	Task 1	Task 2	Task 3
Interrupt During Atomic Task	65%	58%	60%
Interrupt After Atomic Task	89%	90%	92%

environment that we use to develop "Evasive Spectre." We then profile the original Spectre to study its feasibility to perform an attack without detection and discuss strategies to develop an "Evasive Spectre."

3.1 Experimental Setup

We designed an attack on a machine similar to a typical victim laptop computer with Debian Linux 4.8.5 OS on an Intel Core i3-3217U 1.8 GHz processor with 3 MB cache and 4 GB of memory. We used the standard profiling infrastructure on Linux *perf* tools to obtain four performance counters data as discussed in Section 2 including branch mispredictions, LLC misses, branches, and LLC references at each sampling interval.

In the clean environment, we sought to create realistic scenarios by browsing popular websites and streaming videos or running a text editor. For data collection when the system is under attack, we launched Spectre variant 1 proof of concept and "Evasive Spectre" attacks using the strategies proposed in Section 3.3 on top of normal applications. The system status was reset after each run to ensure the measurements were independent across different runs.

Overfitting is a common problem for machine learning classifiers. To allay this problem, we first sought to include more data by collecting data in 10 independent runs and use the same number (1200) of samples from malicious and normal classes. Then, we randomly divided the collected data into training (80 percent) and test (20 percent) data, then separated the training data into training (80 percent) and validation (20 percent) data for cross-validation. We also included regularization parameters in the machine learning models during training.

3.2 Feasibility Analysis of "Evasive Spectre"

In order for the attack to be successful, the attacker has to complete malicious tasks faster than the detection frequency. Thus, the microarchitectural trace of the attack should be reshaped so the attack can make progress at each detection interval. We thus define an *atomic task* as a sequence of instructions that should not be interrupted during execution if progress is to be made towards the completion of a malicious task to achieve successful attack. We identified three *atomic tasks* in the proof of concept Spectre-V1 [1]: *(1) Flushing cache lines, (2) Mistraining branch predictor, (3) Attempting to infer the secret byte that is loaded into cache.* We compare the attack success rate of interrupting the attack during atomic tasks and between atomic tasks by inserting the same instructions. Table 1 shows that the attack success rate drops significantly when defined atomic tasks are interrupted. Therefore, we choose to reshape the microarchitectural features for evasion at the atomic task level rather than a finer granularity in the control flow graph. We profiled the attack on the machine mentioned in Section 3.1. The three *atomic tasks* respectively take 10 μs, 13 μs, 38 μs to complete on average. For each byte, the three tasks were performed multiple times to get the best results. The original attack read secret bytes at an approximate rate of 2 KB/second on average.

As discussed in Section 2, we assumed the attacker knows the features being monitored but does not know the classification period. we used the method proposed in [10], to collect multiple pairs of testing and training data sets of the same features using different collection periods and train a reverse-engineered detector for each data set. The victim's collection period (100 ms) is the same as the collection period of the reverse-engineered detector with the highest accuracy.

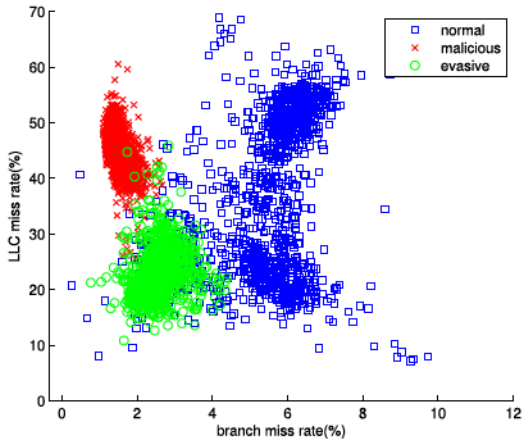

Fig. 1. Branch miss rate versus LLC miss rate for evasive spectre.

Since the sampling period of the victim detector is much larger than the time taken to perform each *atomic task*, the attacker can transform the microarchitectural profile of Spectre by inserting instructions or "sleeping" at a finer granularity than the sampling rate of the detector. To analyze the feasibility of evading detection, we execute *atomic tasks* using 20 percent of each sampling period and put the attack to sleep for the remainder. Fig. 1 shows the distribution of (1) benign, (2) malicious and (3) evasive sample points using cache miss rate and branch miss rate features. It shows a clear boundary between normal and malicious sample points while the evasive sample points shift the original malicious to overlap with normal ones (they cannot fully overlap because unlike normal programs, the evasive attack still needs to perform malicious tasks). Modified Spectre is performs with an 89 percent success rate. This shows the feasibility of constructing an "Evasive Spectre."

3.3 Strategies to Construct Evasive Spectre

As discussed in [5], the original Spectre increases LLC misses and reduces branch mispredictions. To evade detection, the attack could be slowed by putting it to sleep or inserting instructions that reduce the number of LLC misses (reading the same memory bytes) and increase the number of branch mispredictions (adding unpredictable branches). Assuming the attack runs in a loop, at each cycle, the attacker needs to complete a series of atomic attacks to retrieve one secret byte from the victim. We thus considered the following four strategies:

1) Put the attack to sleep in between *atomic tasks*.
2) Put the attack to sleep after all tasks have completed.
3) Insert instructions in between *atomic tasks*.
4) Insert instructions after all tasks have completed.

The first two strategies slow down the attack and strategies 3 & 4 directly manipulate the performance counters.

Varying sleep time or looping the instructions that reshape the microarchitectural profile different times will accordingly change the attack bandwidth. We studied the effectiveness of different strategies by gradually reducing the attack bandwidth and analyzing the results in the following section.

4 Experimental Results

We now evaluate different evasion strategies proposed in Section 3.3 by varying the bandwidth reduction from 1X to 7X and comparing the detection accuracy and the attack success rate. We define the success rate as the percentage of correct bytes inferred by the attacker over the total number of bytes inferred.

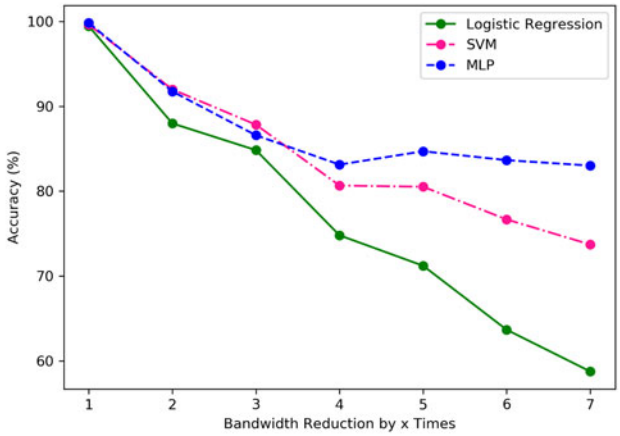

Fig. 2. Detection accuracy - strategy 1 (sleep after all tasks).

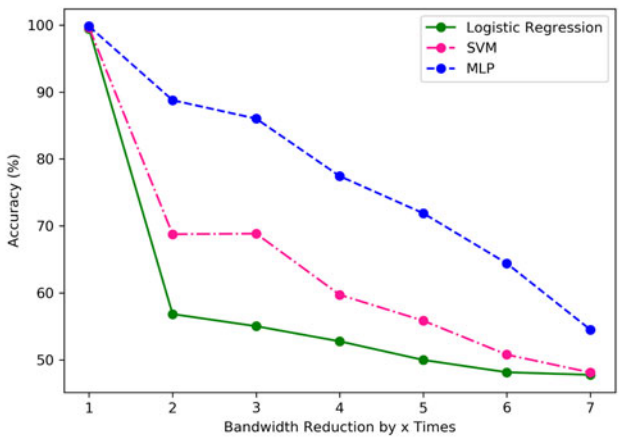

Fig. 3. Detection accuracy - strategy 2 (sleep between atomic tasks).

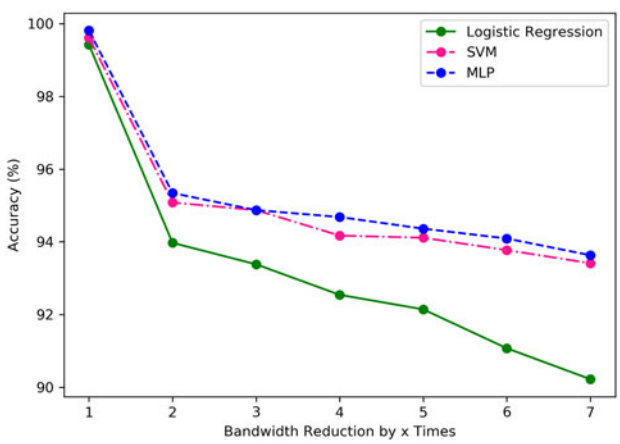

Fig. 4. Detection accuracy - strategy 3 (insert instructions after all tasks).

Putting the attack to sleep or inserting instructions to reshape the microarchitectural profile of the attack reduces the rate of confidential content read. The longer the attack takes to reshape the profile, the more bandwidth reduction it will incur and the lower the success rate, due to higher TLB and cache pollution. Therefore, an effective evasion strategy should result in a low detection accuracy and maintain a reasonable attack success rate and bandwidth.

For each experimental setup with different evasion strategies and bandwidth reduction, we record the attack success rate and detection accuracy using the existing victim detector with different

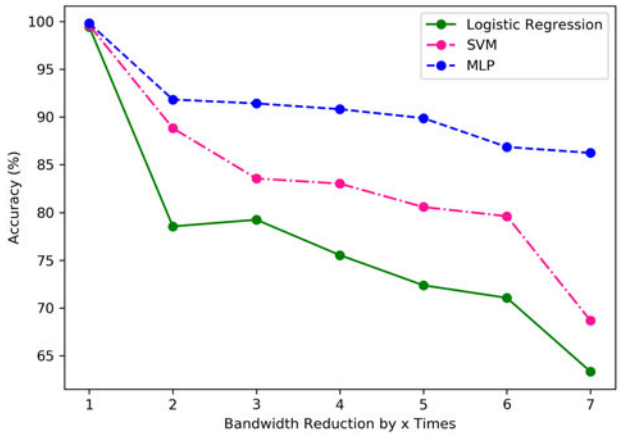

Fig. 5. Detection accuracy - strategy 4 (insert instructions between atomic tasks).

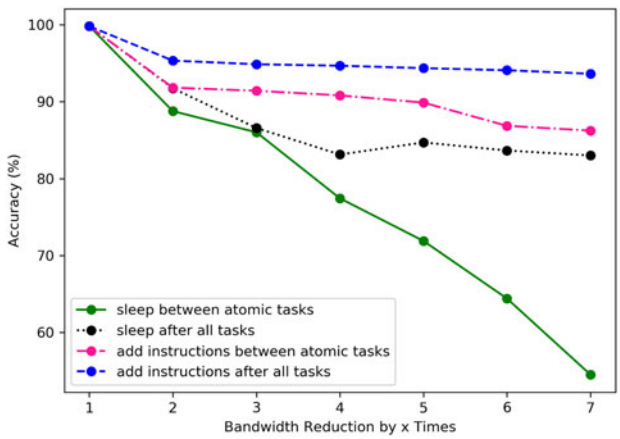

Fig. 6. Detection accuracy using multi-layer perceptron.

Machine Learning classifiers (Logistic Regression (LR), Support Vector Machine (SVM), and Multi-Layer Perceptron (MLP)). We collect data in 10 independent runs for each setup and calculate the average values to avoid bias.

Figs. 2, 3, 4 and 5 show detection accuracy versus bandwidth reduction using different ML classifiers for each of the four strategies. In all cases, the detection accuracy drops with the bandwidth because the attack becomes more evasive and closer to benign programs as it runs slower. In addition, the MLP classifier retains a better detection accuracy as the bandwidth drops. Therefore, MLP has a higher resiliency to evasive attacks. In contrast, it is easier to avoid detection by a simple LR classifier.

Fig. 6 show the detection accuracy of different evasion strategies as bandwidth decreases using MLP. Strategy 1 causes the detection accuracy to drop fastest as bandwidth drops. The detection accuracy diminishes to around 50 percent (random guess) when the bandwidth reduction is 7X. Therefore, strategy 1 produces the most evasive attack. On the other hand, strategy 4 performs the worst in this regard. Strategy 2 and 3 perform similarly in terms of evading detection. Note that shaping the microarchitectural profile in between atomic tasks yields a more evasive attack than shaping it after all the tasks are done no matter what method (sleep or insert instructions) is used.

Fig. 7 shows the attack success rate using different evasion strategies. As the attack bandwidth decreases, so does the success rate. Therefore, an attack is more likely to fail when it is running more slowly due to possible TLB and cache pollution by other processes. Similarly, inserting instructions or sleeping between atomic tasks has a higher chance to fail than inserting or sleeping after all tasks

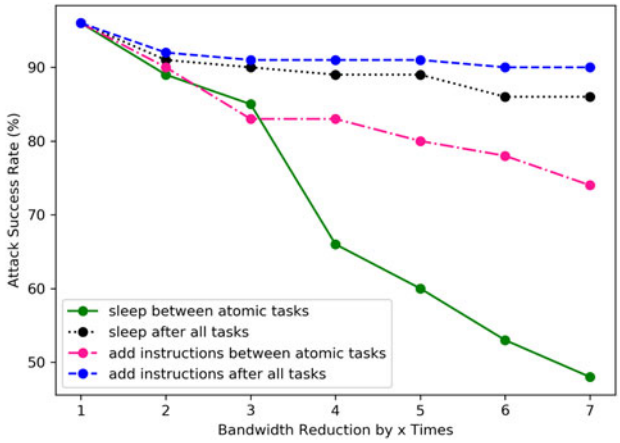

Fig. 7. Attack success rate using the proposed evasion strategies.

are done. The success rate drops below 50 percent with strategy 1 at 7X bandwidth reduction.

To better evade detection (or to reduce detection accuracy), the attack must run at lower bandwidth and success rate. Considering the trade-off between detection accuracy and attack success rate, strategy 1 produces the most evasive attacks but the lowest success rate. Conversely, strategy 4 has the best success rate but is the least evasive. Strategy 2 gives slightly more evasive attacks than strategy 3 as bandwidth reduces further; strategy 2 has a better attack success rate than strategy 3. Therefore, strategy 2 is on the overall most effective. At a 7X bandwidth reduction, the LR classifier can only perform at 58.77 percent accuracy, i.e., no better than a random guess. Meanwhile, the attack success rate remains at 85 percent. Therefore, with strategy 2, the attacker can evade detection by an LR classifier without sacrificing much in terms of success rate and bandwidth. To evade the scrutiny of a more complex classifier such as MLP, the attacker can further reduce the bandwidth for a lower detection accuracy. At 10X bandwidth reduction, the MLP classifier performs at 70 percent accuracy and the attack success rate remains at 85 percent.

5 Conclusions and Future Research

We have demonstrated the feasibility of a re-written Spectre which evades HPC-based malware detectors and proposed evasion strategies including putting attacks to sleep and inserting instructions. We have shown that putting Spectre to sleep after it has performed malicious tasks allows an attacker to effectively evade simple LR malware classifiers and maintain as high a success rate as 86 percent with a concomitant 7X bandwidth reduction. Complex models such as MLP mean higher resiliency to evasion, however, with further bandwidth reduction at 10X, the detection accuracy reduced to 70 percent.

More sophisticated detectors using other ML models such as Recurrent Neural Network with more features for different variants of Spectre will be studied. They can be used to counter evasion by using a higher or randomized sampling rate. Also, a strategy to reduce the performance overhead of the detector such as using dedicated hardware to implement the malware classifier will be explored. As malware becomes pervasive and stealthier, future security mechanism should actively monitor even the subtlest anomalies or signs of malware infection in every layer of the system from networks, software applications to hardware. The defense system should also be able to proactively respond to and remedy threats and be easily reconfigured to future attacks.

Acknowledgments

This work was supported in part 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.

References

[1] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," in *Proc. 40th IEEE Symp. Security Privacy*, 2019, pp. 1–19.
[2] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *Proc. 40th Annu. Int. Symp. Comput. Architecture*, 2013, pp. 559–570.
[3] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Research in Attacks, Intrusions and Defenses*. Berlin, Germany: Springer, 2014, pp. 109–129.
[4] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Proc. 18th Int. Symp. Res. Attacks Intrusions Defenses*, 2015, pp. 3–25.
[5] C. Li and J.-L. Gaudiot, "Online detection of spectre attacks using microarchitectural traces from performance counters," in *Proc. 30th Int. Symp. Comput. Architecture High Perform. Comput.*, Feb. 2019, pp. 25–28.
[6] C. Li and J.-L. Gaudiot, "Detecting malicious attacks exploiting hardware vulnerabilities using performance counters," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf.*, Jul. 2019, vol. 1, pp. 588–597.
[7] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Proc. 9th Int. Conf. Recent Advances Intrusion Detection*, 2006, pp. 226–248.
[8] D. Bruschi, L. Cavallaro, and A. Lanzi, "An efficient technique for preventing mimicry and impossible paths execution attacks," in *Proc. IEEE Int. Performance Comput. Commun. Conf.*, 2007, pp. 418–425.
[9] M. Kayaalp, T. Schmitt, J. Nomani, D. Ponomarev, and N. Abu-Ghazaleh, "Scrap: Architecture for signature-based protection from code reuse attacks," in *Proc. IEEE 19th Int. Symp. High Perform. Comput. Architecture*, 2013, pp. 258–269.
[10] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu, "RHMD: Evasion-resilient hardware malware detectors," in *Proc. Annu. Int. Symp. Microarchitecture*, 2017, pp. 315–327.