Detecting Spectre Attacks Using Hardware Performance Counters

Congmiao Li, Member, IEEE and Jean-Luc Gaudiot, life fellow, IEEE
IEEE Transaction on computers, 6 June 2022.

Problem Addressed in this Paper:

Now days, Information is everything which means it is an important to protect information from access by unauthorized parties. Information can be leaked to unprivileged parties through unintended side channels. These unintentional side channel can be timing information, power consumption information, electromagnetic radiation, light emision or sound. By measuring and analyzing the difference in side channel information, confidential data could be discovered by adversaries.

Spectre attacks exploits speculative execution to leak confidential information through unintended side channels by tricking the processor into taking carefully designed malicious code. Speculative execution reduces the processor idle time to improve performance by executing a predicted path before the actual path is confirmed. However, this may leave observable side effects that allow attackers to obtain confidential data. The branch predictor guesses which branch is more likely to be taken according to the execution history. The processor fetches the associated data and instructions. Then it starts speculatively executing them. If the branch predictor guessed correctly, the processor continues running without delay. Otherwise, the processor will roll back the execution state. Instructions only retire after the correct path is known. However, changes to other microarchitectural features such as cache contents are not reversed and it is this information that an attacker will seek to harvest.

In this Paper, the aim to develop an online detector to detect sectre during the attack and improve its resilience to evasion. They use low-level microarchitectural features which can be extracted from HPC to detect the original spectre variant 1. We then study how the original Spectre could be even more maliciously updated to operate effectively without being detected by HPC-based classifiers in a quantitative way.

Motivation for the Problem:

Spectre attacks have different variants and works on various Intel, AMD and ARM platforms. Therefore, there is no patch for the whole class of attacks. To completely solve the problem, changes to processor design and instruction set architectures are required.

Motivation for the Solution:

Instead of changes to processor design and instruction set architectures we can detect spectre attacks and apply mitigation technique. To detect spectre attack, we can use low level hardware microarchitectural features extracted from HPCs and apply Machine Learning Model to classify it to binary 0 and 1. 0 means no attack and 1 means there is attack going on. If model classify it to 1 then we can terminate that application and take some precautions.

Theory:

All the variants of spectre attack follow the same principal in that they rely on the changes in the state of the cache caused by the speculative execution. Different variants exploit different kinds of speculative execution following different control or data misprediction, but in the final phase, secret information is usually leaked through a cache side channel. Therefore, we can monitor the cache behaviour to detect the attacks. Their proposed detection approach periodically collected miroarchitectural features from performance counters. They chose to monitor four events related to cache miss rate and branch misprediction rate.

- Cache references
- Cache misses
- Branch Instructions retired
- Branch mispredictions

Machine Learning can be used to train classifiers that determine the class to which a given data set belongs. They used supervised learning to train the attack detector with a set of pre-labeled exapmles. They chose three different machine learning algorithm including Logistic Regression, Support Vector Machine and Artificial Neural Networks. They used a sliding window-based online classification methods to detect malicious behaviour at runtime. To this end, we collected microarchitectural features at a 100 ms sample period. The multidimensional data were then fed to a machine learning classifier to make decisions as to whether malicious code was being executed or not. The problem of detecting malicious attacks in real time was to make decisions according to the binary time series generated by the base classifier.

Evasive Spectre-

How the original spectre could be even more maliciously updated to operate effectively without being detected by our proposed detector. To achieve reasonable attack success rates, the attacker has to insert instructions or put the attack to sleep at a coarser granularity than a basic block in the control flow graph. To achieve the goal, we assume the attacker can observe the behaviour of the malware classifier from a machine with a similar HPC-based detector as the victime machine. The attacker can evade detection by changing the microarchitectural characteristics of the updated spectre so as to behave like a benign program.

The attacker knows the features being monitored by the malware classifier. However, the attacker does not know the sampling period of the detector. Yet this can be reverse engineered. In doing so, the attacker would have to sacrifice the efficiency of the attack and perform it more slowly. Slowing down teh spectre could result in failure of the attack. In order for the attack to be successful, the attacker must complete malicious tasks faster than the detection frequency. We thus define an atomic task as a sequence of intructions that should not be interrupted during execution if progress to be made towards the completion of a malicious task to achieve success. 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. The original spectre increases LLC misses and reduces branch mispredictions. To evade detection, the attack could by slowed by putting it to sleep or inserting instructions that reduce the number of LLC misses and increases the number of branch mispredictions.

How to imrove the spectre detector to be evasion-resilient by randomly switching between multiple detectors with different settings. Randomization introduces errors to reverse-engineering and makes the detector resistant to evasion. They build detectors with different settings in terms of features used for detection and sampling periods and randomly switch between them with equal probability. The baseline detector consist of only one detector which uses a combination of cache and branch related features with a fixed sampling period of 100ms. Then they build 3 detectors with randomized features to be either cache related, branch related or the combined features and switch between them in an upredictable order. Finaly, they build 6 detectors by randomizing both features and sampling periods. However, for increased diversity with randomized features and periods, the reverse-engineereing failed to infer the accurate settings of the victim.

Implementation:

They ran the attack on typical personal laptop 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 DDR3 memory from micron. They used the standdard profiling infrastructure of Linux, perf tools to obtain performance counter data. Ran perf in sampling mode and record the measurement every 100ms so as not to excessively degrade the performance. They collected system wide data rather than for individual processes. To experiment with systems under high load conditions, besides running the above-mentioned applications, They stressed the system with a memory intensive load running *memcpy*, a CPU intensive load running floating-point operation *cfoat* in a loop and both memory and CPU intensive workloads running in parallel. The data was collected from a "clean" environment where the computer runs typical desktop applications such as web browser, video player, and text editors, as well as from an environment under Spectre attack while running the same

desktop applications. The data was then labeled to train the machine learning classifier to classify the input data at each time interval. At runtime, the output time series from the trained classifier was fed into the online detection mechanism to decide if the system was under attack.