Detecting the Exploitation of Hardware Vulnerabilities using Electromagnetic Emanations

Giancarlo Canales Barreto¹ Nicholas Lamb²

Abstract—We present a cache attack monitoring methodology that leverages statistical machine learning models to detect n-day hardware attacks by analyzing the electromagnetic emanations of a device. Experimental results from a Raspberry Pi 4 hosting Linux and a Jetson TX2 development board running a Linux guest hosted by seL4 demonstrate that our approach can sense Spectre attacks with a concordance statistic of 97% and 95%.

Index Terms—cache attacks, intrusion detection, electromagnetic emanations, side-channel analysis, signal processing and analysis

I. INTRODUCTION

Modern microprocessors employ various optimization techniques such as caching and pipelining. Unfortunately, while these methods boost performance they also increase the microprocessor’s complexity and open the door to unintended operations that can break hardware security policies. Well known examples of attacks that leverage these problems include Spectre [1] and Meltdown [2], which can be used to access otherwise-unavailable memory and information by exploiting cache timing side-channel leakages.

Timing side-channel attacks can be realized using cache attack primitives. Examples include Evict+Time and Prime+Probe to conduct attacks against OpenSSL [3], Flush+Reload to perform a high-resolution attack against GnuPG [4], Flush+Flush to implement a stealthy cache attack against AES [5], and Prime+Abort to extract cryptographic keys using transactional memory extensions [6]. In turn, these primitives can be further broken down to: Evict, where cache data is replaced with new data; Time, where the amount of time it takes an operation to complete is measured, Prime, where a special condition within the system is triggered; Probe, where cache lines that were used are identified; Flush, where the cache is cleared; Reload, where the cache data is reloaded; and Abort, where a dummy transaction is initiated and eventually canceled.

These primitives can also be used to implement memory corruption attacks, and can be considered a type of cache-like attack. One notable example is Rowhammer [7] which uses the Flush+Reload primitives to attack Dynamic Random-Access Memory (DRAM) implementations such as Double Data Rate 3 (DDR3) DRAM. Additionally, covert communication channels that use any of these primitives can also be included in this category.

A. Related Work

Cache attacks can be detected by using software that analyzes performance counters [8], [9] in the microprocessor. This approach is possible because these attacks are repetitive, and in some cases can take days to successfully execute, as is the case with the ECCPloit [10] attack. An alternative approach is to employ software mitigations which can result in performance degradation [11].

Leveraging physical side-channel leakages to detect anomalous activity has also been explored by other researchers. Examples include leveraging electromagnetic signals to detect ransomware attacks in cyber-physical systems [12], analyzing and classifying malware in embedded systems [13], and detecting anomalies in medical devices [14].

B. Our Contribution

The contribution made in this paper is a cache attack monitoring technique that leverages statistical machine learning models to detect n-day hardware attacks. More precisely, we demonstrate how the Flush primitive can be sensed using the radio-frequency emanations of a microprocessor and leverage this phenomenon to detect a variant of Spectre that utilizes the Flush primitive. Two detection experiments are presented in this paper. The first experiment targets the Jetson TX2 board while it runs a Linux guest that is hosted by the seL4 microkernel hypervisor, and the second experiment utilizes a Raspberry Pi 4 Model B with Linux.

II. METHODOLOGY

A. Experiment Setup

The two experiments presented in this paper were conducted utilizing the configuration shown in Figure 1. Both experiments utilize the Aariona MDF 9400 antenna that sits below a Device Under Test (DUT), and both are contained within a RF shielded enclosure. The antenna is connected to the HackRF One Software Defined Radio (SDR). Signals acquired by the SDR were sampled with an instantaneous bandwidth of 20 MHz centered at 10 MHz, and forwarded to a Linux workstation for analysis.

B. Flush Primitive

The Flush cache-attack primitive can be implemented in x86_64 processors using the CFLUSH instruction. In AArch64 processors the primitive can be implemented using the DC CIVAC (Data Cache maintenance, Clean Invalidate by Virtual Address to the point of Coherency) and DSB SY (Data...
RF Shielded Enclosure

Device Under Test

Antenna

Software Defined Radio

Signal Processing System

Fig. 1. The experiment setup used in this paper. Signals emanated by the Device Under Test (DUT) are received using a magnetic directional antenna and digitized with a Software Defined Radio (SDR).

Synchronization Barrier, full SYstem) instructions. The pseudocode of the Flush primitive is shown in Algorithm 1. The DC CIVAC instruction will invalidate a cache line asynchronously, and to ensure completion of this routine, the DSB SY instruction is needed to synchronize execution until the maintenance routine finishes.

Algorithm 1 Flush Primitive

Input: Buffer (256 × 512)
1: function FLUSHALL(Buffer)
2: Address ← GETADDRESS(Buffer)
3: for off = 0 to 255 do
4: Invalidate(Address + [off × 512])
5: SYNCHORNIZE()
6: end for
7: end function

C. Training Program

The Flush primitive for AArch64 can be implemented using DC CIVAC and DSB SY, and is used to create two training programs. Each program is executed serially and the corresponding radio-frequency emanations are repeatedly captured and stored as training data for our statistical machine learning model.

Algorithm 2 Training Program

Input: Buffer (256 × 512)
1: while True do
2: Buffer ← INITIALIZE()
3: OPERATION(Buffer)
4: end while

To prevent compiler optimizations that would yield unwanted differences in the training program, the pseudocode shown in Algorithm 2 was compiled into intermediate assembly, and split into two programs that include only one of the cache instructions, where OPERATION can either be DC CIVAC or DSB SY.

D. Operation Frequency Response (OFR)

Signals are obtained using the HackRF One with a center frequency of \( f_c = 10 \) MHz and a sampling rate of \( f_s = 20 \) MS/s. The signals are stored as In-Phase/Quadrature (IQ) data, a type of analytic signal that is represented using complex numbers. About 20 seconds of data is acquired for each training program, sliced into 1 ms trace windows, and normalized so that the mean is \( \mu = 0 \) and standard deviation is \( \sigma = 1 \).

\[
P_t[k] = 10 \cdot \log_{10} \left| \sum_{n=0}^{N_t-1} x[n] \cdot e^{-j2\pi kn} \right|^2
\]

After the signals are sliced and normalized, the Power Spectral Density (PSD) of each trace \( x_t[n] \), where \( t \) is the traces index, is estimated using the discrete Fourier Transform (DFT) shown in (1). The absolute value of a complex number is defined as \( |a + bi| = \sqrt{a^2 + b^2} \). Additionally, the average power spectral density of the traces is computed as shown in (2), where \( N_t \) is the number of traces.

\[
P[k] = \frac{1}{N_t} \sum_{t=1}^{N_t} P_t[k]
\]

The resulting PSD is referred to as the Operation Frequency Response (OFR) and is used to sense the execution of DC CIVAC and DSB SY. Only the DC CIVAC OFR is used to detect the execution of the Flush primitive.

E. Template Analysis

A statistical machine learning technique called template analysis [15] is used to determine if an instruction is being executed or not. This approach utilizes the multi-variate Gaussian Probability Density Function (PDF). An example 3-variable Gaussian PDF is shown in (3), and the corresponding covariance matrix \( \Sigma \) and mean vector \( \mu \) are shown in (4) and (5), where the variables \( f_0, f_1 \) and \( f_2 \) are random variables.

\[
f(x) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)
\]

The random variables \( f_0, f_1 \) and \( f_2 \) correspond to three distinct points of interest within the OFR. That is, three different frequency offsets that encode information unique to the instruction we are attempting to detect. These points are selected by measuring the absolute sum of differences between the OFR of DC CIVAC and DSB SY. Additional instructions may be evaluated using this approach to develop a m-ary detection model, but for the purpose of this paper, only two instructions are chosen. Frequencies with the highest power difference are selected as the points of interest.

\[
\Sigma = \begin{bmatrix}
\text{Var}(f_0) & \text{Cov}(f_0, f_1) & \text{Cov}(f_0, f_2) \\
\text{Cov}(f_1, f_0) & \text{Var}(f_1) & \text{Cov}(f_1, f_2) \\
\text{Cov}(f_2, f_1) & \text{Cov}(f_2, f_1) & \text{Var}(f_2)
\end{bmatrix}
\]
In this paper, the PDF of an OFR is called the “template”. The template can be used to determine the probability that a particular instruction is executing by feeding it an OFR’s points of interests.

F. Spectre Attack

The variant Spectre attack detected in our experiments relies on conditional branch misprediction. The pseudocode for the attack primitive is shown in Algorithm 4.

Algorithm 3 Conditional Branch (Prime)

1: function VICTIMFUNCTION(x)
2:  if \( x < 16 \) then
3:    Data \( \leftarrow \) Array2[Array1[x] \times 512]
4:    Temp \( \leftarrow \) Temp AND Data
5:  end if
6: end function

The Prime primitive can be used to mount an attack in three phases. The first phase consists of invalidating all cache lines using the Flush primitive, followed by triggering branch misprediction using the Prime primitive, and finally measuring the access time of each cache line using Probe primitive. A generalized view of each phase of the attack is shown in Algorithm 4 [16].

Algorithm 4 Spectre Attack

1: while Attacking do
2:    FLUSHALL(Array2) \( \leftarrow \) Flush
3:    VICTIMFUNCTION(...) \( \leftarrow \) Prime
4:    TIMEDREAD(Array2) \( \leftarrow \) Probe
5: end while

G. Spectre Detection

The DC CIVAC template is used to detect the execution of a Flush primitive and is consequently used to determine if a Spectre attack is being conducted.

Additionally, two programs are used to demonstrate detection. The first program lacks any attack, and may be a stress test or a version of Spectre that lacks cache instructions and fails to succeed, whereas the second program is a Spectre attack that includes cache instructions and successfully attacks the system.

Finally, the OFR for the execution of each program is estimated and fed to the templates. We emphasize that the model used to detect Spectre was trained using the Flush OFR instead of the actual Spectre program. The null and alternate hypothesis for all experiments are: \( H_0 \) system is not being attacked, \( H_1 \) a Spectre attack against the system is underway.

III. RESULTS

Results presented in this section indicate that the Flush OFR template classification method generalizes quite well and can be used to detect the Spectre attack on AArch64 microprocessor systems.

A. Jetson TX2

In this experiment the DUT is a Jetson TX2 (ARM Cortex-A57) that is running a Linux guest that is hosted by seL4. The \( H_0 \) hypothesis is a stress test that exercises I/O, memory, and CPU funcions, and \( H_1 \) is a successful Spectre attack. This scenario was chosen to test the OFR’s ability to not falsely detect the execution of spurious DC CIVAC instructions during simulated stress loads within a virtualized environment.

The average Flush and DSB SY OFR for the Jetson TX2 device is shown in Figure 2. It is interesting to observe that there appear to be some evenly spaced harmonics associated with the operations.

Differences across the frequencies between OFRs are shown in Figure 3 and the top three frequency offsets with the highest power difference are highlighted with a red marker. There appear to be spurs of energy at multiples of 1 MHz, we hypothesize each one of these spurs is being generated by interactions between the loops used to implement the flush operation and the scheduling algorithm [17] employed by the Linux operating system of the DUT. Both of these mechanisms can be seen as oscillators that run at a frequency that is much lower than the processor’s clock. This phenomenon can be seen as an unintentional software defined radio that is modulated by processor instructions.

The specificity and sensitivity of our templates when it comes to detecting Spectre are characterized using a Receiver Operating Characteristic (ROC) curve and is shown in Figure 4. The ROC curve suggests that we can detect Spectre with a concordance statistic of 95% when evaluating a 10ms window using the Flush template. The model for DSB SY was not used to perform detection in our experiments. A confusion
Fig. 3. Points of interest in the instructions used to implement the Flush primitive on the Jetson TX2 system.

Fig. 4. Receiver Operating Characteristic (ROC) curve for Spectre detection in the Jetson TX2 system using the Flush template. The Area Under the Curve (AUC) for each template is also shown.

The normalized confusion matrix for the 10ms detector for the Jetson TX2 system is shown in Figure 5. Detection experiments were done while the DUT was operating under optimal conditions.

B. Raspberry Pi 4

In this experiment the DUT is a Raspberry Pi 4 (ARM Cortex-A72) that is running Linux. The $H_0$ hypothesis is an unsuccessful Spectre attack lacking the Flush primitive instructions, and $H_1$ is a successful Spectre attack. This scenario was chosen to test the OFR’s ability to detect a needle in the haystack, that is, determining how sensitive the model is to the DC CIVAC instruction.

The average DC CIVAC and DSB SY OFR for the Raspberry Pi 4 device is shown in Figure 6. The top three frequency offsets with the highest power difference are highlighted with a red marker in Figure 7.

Fig. 5. The normalized confusion matrix for the 10ms detector for the Jetson TX2 system.

Fig. 6. Points of interest in the instructions used to implement the Flush primitive on the Raspberry Pi 4 system.

Fig. 7. Points of interest in the instructions used to implement the Flush primitive on a Raspberry Pi 4 system.
The ROC curve for the Raspberry Pi 4 experiment in shown in Figure 8 and suggests Spectre with a concordance statistic of 97% when evaluating a 10ms window using the Flush template, and the confusion matrix for the 10ms detector is shown in Figure 9.

An alternative view based on the data used to generate the ROC curve is presented in this paper may be extended to support additional attacks, architectures, instructions, and primitives, including potentially detecting previously unseen zero day attacks since the model is built on operations that are vulnerability agnostic.


text continues...
“Measuring the impact of spectre and meltdown,” in 2018 IEEE High Performance extreme Computing Conference (HPEC), pp. 1–5, 2018.

[12] N. Sehatbakhsh, A. Nazari, M. Alam, F. Werner, Y. Zhu, A. Zajic, and M. Prvulovic, “Remote: Robust external malware detection framework by using electromagnetic signals,” IEEE Transactions on Computers, vol. 69, no. 3, pp. 312–326, 2020.

[13] N. Chawla, H. Kumar, and S. Mukhopadhyay, “Machine learning in wavelet domain for electromagnetic emission based malware analysis,” IEEE Transactions on Information Forensics and Security, vol. 16, pp. 3426–3441, 2021.

[14] N. Sehatbakhsh, M. Alam, A. Nazari, A. Zajic, and M. Prvulovic, “Syndrome: Spectral analysis for anomaly detection on medical iot and embedded devices,” in 2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), pp. 1–8, 2018.

[15] S. Chari, J. R. Rao, and P. Rohatgi, “Template attacks,” in Cryptographic Hardware and Embedded Systems - CHES 2002 (B. S. Kaliski, ç. K. Koç, and C. Paar, eds.), (Berlin, Heidelberg), pp. 13–28, Springer Berlin Heidelberg, 2003.

[16] V. V.E.O, “Poc/cve-2017-5753.” https://github.com/V-E-O/PoC/tree/master/CVE-2017-5753, 2018.

[17] J.-P. Lozi, B. Lepers, J. Funston, F. Gaud, V. Quémé, and A. Fedorova, “The linux scheduler: A decade of wasted cores,” in Proceedings of the Eleventh European Conference on Computer Systems, EuroSys ’16, (New York, NY, USA), Association for Computing Machinery, 2016.