Securing IoT/IIoT from Software Attacks Targeting Hardware Vulnerabilities

Nikolaos Foivos Polychronou, Pierre-Henri Thevenon, Vincent Beroulle, Nikolaos Polychronou, Maxime Puys

Securing IoT/IIoT from Software Attacks Targeting Hardware Vulnerabilities

Nikolaos Foivos POLYCHRONOU[1], Pierre-Henri THEVENON[1], Maxime PUYS[1], Vincent BEROULLE[2]

[1]*Univ. Grenoble Alpes, CEA, LETI, DSYS*, Grenoble, France
Firstname.Name@cea.fr
[2]*Univ. Grenoble Alpes, Grenoble INP, LCIS*, Valence, France
Firstname.Name@lcis.grenoble-inp.fr

***Abstract*—The microarchitecture of modern systems become more and more complicated. This increasing complexity gives rise to a new class of attacks which uses software code and targets hardware vulnerabilities of the system microarchitectures. Software attacks targeting hardware vulnerabilities (SATHVs) gain popularity. In particular, cache side channel attacks, Spectre, and Rowhammer are serious threats. They take advantage of microarchitectural vulnerabilities to extract secret information or harm the system. As these attacks target the system's hardware, they can avoid traditional software antivirus protections. However, they modify the normal operation of the system's hardware. Hardware Performance Counters (HPCs) are special registers that allow counting specific hardware events. These registers can help us monitor system's execution at hardware level and detect this set of attacks. Many solutions in the literature use HPCs to detect SATHVs. Although, these solutions target detecting only a limited set of the available SATHVs. If security designers do not consider all the possibilities, attackers can bypass existing protections using SATHV variants. In this article, we investigate how the side effect selection proposed in the literature, could or could not help us detect the studied attacks in our testing platform. Our threat model includes Cache side channel and Rowhammer attacks. We also discuss the limitations of software monitoring and how hardware approaches can resolve them.**

***Index Terms*—Security, Hardware performance counters, Attacks, Malware, Microarchitecture, Detection**

I. Introduction

There is an increasing use of Internet of Things (IoT) and Industrial IoT (IIoT) devices. The amount of sensitive information they handle, the minimal protections integrated due to a desired low cost, and the increasing complexity of modern microprocessors requires to increase the security of these devices. This is supported by recent cyber-security threats such as Stuxnet and Mirai. Software and hardware attacks have been studied for decades. However, a new class of attacks which we refer to as Software Attacks Targeting Hardware Vulnerabilities (SATHVs) arise. These attacks target vulnerabilities in the microprocessor hardware architecture. They require no physical access, in opposite to traditional hardware attacks, and they behave as normal applications. Thus, they are a great security threat. SATHVs target both the microarchitecture hardware vulnerabilities and side-channel leakages. They allow the attackers to extract secret information, implant malicious code, or gain access to privileged code.

a) Related works: Such attacks can hardly be detected by antivirus programs because they target hardware security vulnerabilities and because they leave no traces in traditional log files. To address this, researchers propose the use of Hardware Performance Counters (HPCs) for online detection. HPCs are special purpose registers, which count hardware related events in the microprocessor. In 2013 [1], Demme et al. show that it is possible to detect malware using HPCs. In 2016 [2], Payer proposed mechanism allowing to detect some Cache Side Channel attacks (CSCAs) plus Rowhammer. In 2017 [3], Peng et al. also propose a detection mechanism for CSCAs and Rowhammer.

b) Contributions: In this article, we propose to implement detection mechanisms presented above on our test platform and assess how accurate they are in detecting SATHVs. We show that slight modifications on the experimental setup can drastically decrease their detection rate and we discuss how correct choice of side effects (metrics monitored by HPCs) alongside hardware monitor implementation can help.

c) Outline: In Section II, we describe the methodology we used to replicate state-of-the-art experiments. In Section III, we present our experimental results and discuss the limitations of proposed implementations. Then, in Section IV we discuss how hardware can resolve some of the limitations. Finally, in Section V we conclude and summarize our work.

II. Methodology

As attacks modify the behavior of the microprocessor, we propose an experimental platform allowing us to test and evaluate different side effects. Our goal is to test attack libraries adapted to our target. To find how they differ from normal applications, we choose normal programs representative of the applications running in our target.

Figure 1 describes our methodology. In step 1, we analyze the detection capabilities of our platform. First, we want to know what are the available hardware events. Then, we need to know if access to these monitoring capabilities is allowed. Finally, a set of hardware events is selected for analysis plus the monitoring interval. The monitoring interval specifies the time between consecutive measurements.

Step 2 implements a monitoring module which configures the HPCs with the hardware events selected from Step 1. It resets

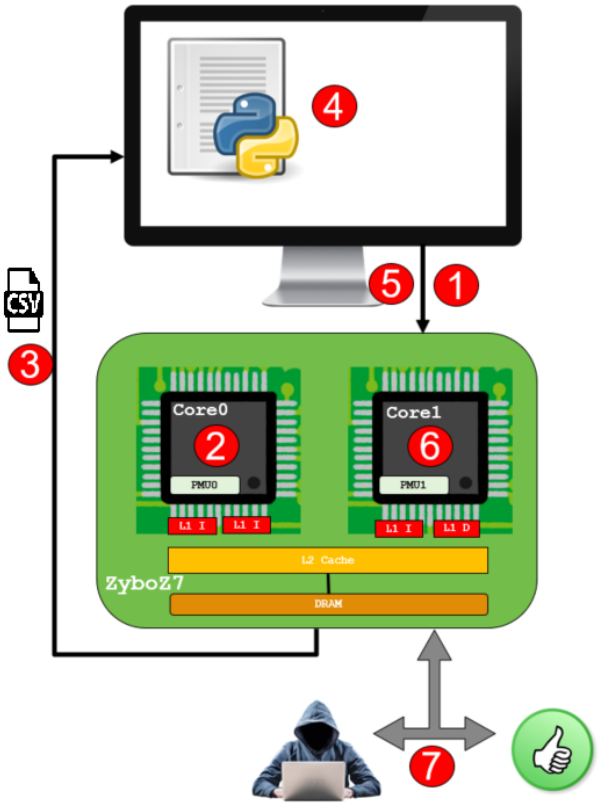


Fig. 1: Methodology

the HPCs and sleeps for the specified monitoring interval letting the application execute. After the specified time, it disables the HPCs. This is necessary to disregard the behavior of our monitoring module from the trace. It reads the counter values and extracts them to a .csv file. Finally, it re-enables the HPCs with the configured events, and continues the same loop until all the application data set executes.

Step 3 and 4 includes the extraction of the raw data traces from the HPCs for offline analysis via a Python script. We plot the raw data traces, boxplots, and distributions of each application. We treat each application separately, or we categorize normal applications with the same label and attacks with different. The offline analysis shows the effectiveness of the side effects selection using our platform and the available attack vectors. The loop of Steps 1, 2, 3, and 4 is repeated for all the available hardware events.

In Step 5, we choose the side effects that gave us the best results during the offline analysis. If there is no clear separation between the behavior of a normal and malicious application, the accuracy of a detection mechanism decreases. Steps 6 and 7 describe how to effectively monitor the application in production after the benchmarking phase described in this paper. Step 6 is the implementation of a detection module using the selected hardware events to monitor. The detection module runs in parallel with the applications, as the monitoring module. In Step 7, instead of exporting the results to a .csv file, the detection module stops the execution of the system, in case of malicious activity detection.

III. Results

The platform that we use for testing is the ZYBO Z7, a Zynq-7000 ARM/FPGA SoC Development Board. It is equipped with a 667MHz dual-core Cortex-A9 processor with 1 GB DDR3L memory. The system is equipped with a Debian GNU/Linux 10. The instruction set is based on ARMV7. For the ARM Cortex A9, there are in total 55 available events. Our platform allows us monitoring 6 events concurrently. In our platform, this is forbidden and a kernel module was written to allow access to HPCs. The aforementioned mechanisms studied attack vectors that take advantage of the existence of the *flush* instruction. As ARMV7 cache maintenance instructions are not available in userspace, we use eviction based techniques to reproduce the studied attacks. Our attack library includes CSCAs in the AES T-Tables and Rowhammer. The CSCA in the AES T-Table uses the Evict+Reload [4] approach. For our normal applications, we use MiBench [5] and PARSEC [6] suites. We choose MiBench suite as it has the goal of representing the spectrum of embedded applications used in the industry. On the other hand, PARSEC focuses on the application domains in financial, computer vision, physical modeling, future media, content based search, and deduplication.

We focus our experiments on [2] and [3]. These solutions have no general fair cost strategy to detect set of multiple attacks, and they need to update the mechanisms depending on the application and the platform. This raises as a major problem. As new attack vectors appear every year, and the big database of existing ones, detection mechanisms are limited to specific attack vector detection and with no possibility to upgrade them in the future to include the new ones, will not be adopted by vendors.

We studied the proposed mechanisms, to understand their limitations and their advantages. Table I summarizes authors' experimental setups with regard to ours.

A. *Effectiveness of proposed solutions on our platform*

Eviction techniques try to remove the targeted address from the cache without the help of cache maintenance instructions. To succeed attackers must find a set of addresses that map in the same cache line. When loaded, they evict the targeted address. Because of the random-replacement policy in ARM caches, more addresses than the cache line size are needed. Eviction is thus slower than just "flushing" an address. This difference greatly modifies the expected side effects. Attackers can be more versatile using different eviction strategies. Figure 2 illustrates this. We can observe that the L2 miss rate and number of L2 misses do not increase for attacks compared to normal applications. Rowhammering using eviction showed to be possible by [11].

In HexPADS [2] the attacks studied were the Flush+Reload, Prime+Probe, and Rowhammer. The side effects measured are the number of executed instructions, Last Level Cache (LLC) accesses and LLC misses. In addition, they use the status information of each process as exported from the kernel, e.g.,

Reference	Platform	Normal applications	Attack library
HexPADS [2]	Intel Core i7-3770 CPU 4 cores, 3.40 GHz, 16 GB, Ubuntu 14.04	SPEC CPU2006 PARSEC 3.0	Rowhammer, Flush+Reload [7] ctemplate [4], C5 [8]
Peng et al. [3]	N/A	Quicksort, urlopen	Rowhammer, Flush+Reload, ctemplate C5, RSA timing attack [9] Web timing attack [10]
Our	Zynq-7000 ARM/FPGA SoC, 667MHz dual-core Cortex-A9, 1GB, Debian GNU/Linux 10	MiBench, PARSEC 3.0	Rowhammer, Evict+Reload

TABLE I: Experimental setup comparison

number of minor page faults, number of major page faults, and execution time. HexPADS detects CSCAs and Rowhammer using three features. If the cache miss ratio is greater than 70%, the total number of cache misses more than 100 million, and page fault miss rate of less than 0.01% then attacks are detected. These metrics are platform dependent. As we can see from Figure 2, the cache miss ratio is not a good indicator in our platform and attack selection. We can see that the average for CacheSCA is at the same level as for normal applications. Rowhammer has on average a higher miss rate, on the other hand. The cache misses decreased for both attacks. This is because during eviction techniques we observe a lot of hits in the cache. Also, eviction is slower than just flushing the targeted address. As one of the indicators used for attack detection by HexPADS is not performing as expected in our platform, a similar detection technique is not applicable.

Peng et al. [3] observed that normal applications can exhibit a high cache miss ratio. To reduce the false positives, they used another indicator regarding the Data Translation Lookaside Buffer (DTLB) miss rate, which set as 0.2%. Figure 2c presents the DTLB miss rate. CSCA and Rowhammer attacks exhibit a reduced DTLB miss rate on average than normal applications. But, as we can see from the boxplot of normal applications, the distribution of measurements is wide. The main reason behind the increase in the expected DTLB miss rate during eviction based attacks is that when the attackers try to find congruent addresses to evict the target, they request data addresses that they did not use before as shown Figure 3. A lot of MiBench applications exhibit low DTLB miss rate and high LLC miss rate.

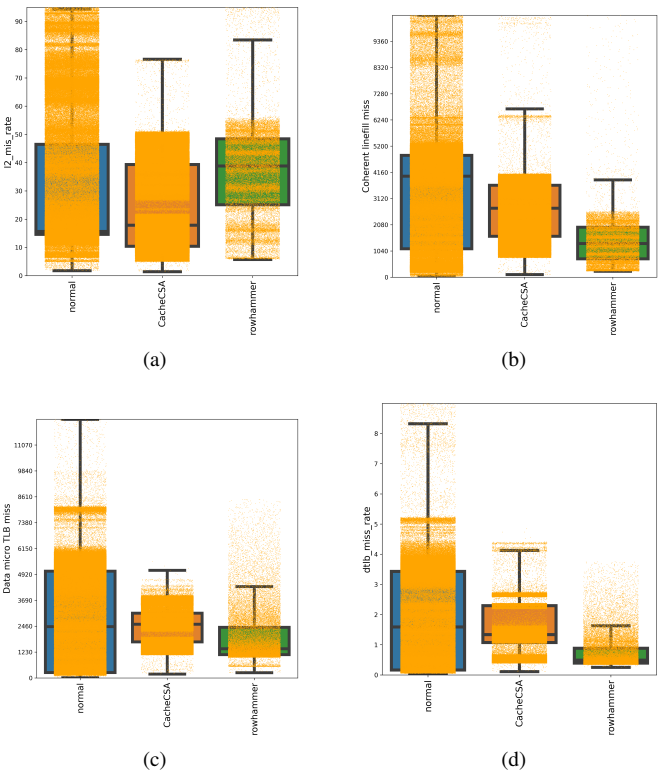
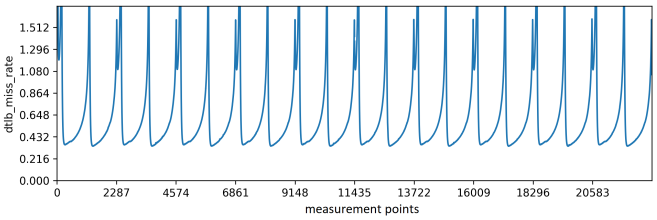


Fig. 2: (a) L2 miss ratio, (b) L2 misses label as Coherent linefill miss, (c) DTLB miss rate, (d) DTLB misses.



(a)

Fig. 3: 10 round Rowhammer DTLB miss rate.

B. *Limitations of existing approaches*

Both of the aforementioned approaches use the utilities of the operating system to extract the information from the HPCs. This limits their ability for better time resolutions. *Perf* tool [12] is a performance analyzing tool in Linux. The minimum time interval between consecutive measurements for *perf* is 100ms. HexPADS [2] uses a 1s monitoring time and also averages 60 measurements for each process. On the other hand, [3] uses the *perf* utility, but the time interval between measurements is not specified. However, monitoring with an interval of 100ms is still not accurate enough since Rowhammer must induce the bitflips in less than 64ms. Attackers can use the rest of the time doing operations that decrease the side effects, thus, to fool the detection mechanism. This can be succeeded by sleeping e.g., for 30ms, which will reduce the cache misses during the monitoring interval. For CSCAs, one round of the attack in the AES T-Table implementation took less than 120ms. Attackers can reduce the time needed, increasing the noise in their measurements. Though, it is necessary to be able to detect the attacks as soon as possible.

Another limitation of the two approaches is the use of hard-

coded values. They are platform dependent and a clever attacker could easily bypass the detection with evasive malware.

IV. Discussion

As we observe from the results presented, the side effects selection plays an important role in the accuracy of a detection mechanism. To find broader applications in different platforms, we must choose side effects with stable behaviors among platforms, and between attack variants. From our conducted experiments, we observe that eviction based attacks exhibit different behaviors than expected from the proposed detection mechanisms and thus they could possibly act unnoticed in our platform. To be able to detect them, we need to look closer at their malicious behavior. In addition, the limited amount of events we can monitor concurrently limits our detection capability. For example, the number of L2 Cache accesses is not available on our platform. To calculate it we can use the information from other hardware events. If the additional events are good indicators to detect the attacks, we will not waste resources. Attacks are versatile, with numerous variants, and if not studied carefully, hypothesized protected target can be vulnerable. This was shown, as in CSCA an attacker who uses eviction might be able to bypass the studied detection mechanisms.

Of course, detection mechanism using collected data from HPCs, will be able to detect some of these software attacks. But, we question if implementing a detection mechanism in software is enough. To detect the attacks we monitor the system every specified time interval. Ideally, we want to monitor closely the applications. For our mechanism, we use the *nanosleep()* function. The *nanosleep* function was quite noisy for timing intervals less than 1ms. As well, the minimum time we could use was 120µs. Noisy timings between measurements add extra noise.

Further, most of the attacks require some kind of privileges to succeed. For example, *pagemap* information, that is used for retrieving the virtual to physical mapping, is restricted in Linux. Moreover, to read the cycle counter in ARM, a kernel module must be written to enable access. Other methods to find this information exist, such as reverse engineering the virtual to physical mapping, but increase the attack steps or the noise. Consequently, this decreases the success probability of an attacker succeeding to extract useful information and increases the probability for a security mechanism to detect the attack. If the attackers can access some restricted information, that will let them proceed with the attack, what holds them from modifying the detection mechanism and taking advantage of it to keep them undetected. This is more serious when considering the numerous bugs found in the software code of applications and the operating system. Besides, as software detection mechanisms and the applications share the same resources, there is a performance overhead. Apart from this, sophisticated malware can detect the presence of a detection mechanism, and hide its activity. On the other hand, a hardware implementation can address the issue of performance overheads, and hide its presence from the user applications. Also, a hardware implementation can access more hardware resources than software. This means, that a hardware implementation can have more information that will let it detect with better accuracy the attacks. But, it comes at the cost of adding a new component in the architecture, which increases the design time, surface, cost, etc. Security designers must perform their vulnerability analysis and decide which implementation is better for their system. This is why, we believe a hardware mechanism, independent of the operating system and the applications, with unrestricted access to hardware information, and no time limit between monitoring times are more suitable.

V. Conclusion

In this article, we propose to implement existing detection mechanisms on our test platform and assess how accurate they are in detecting SATHVs. We show that slight modifications on the experimental setup can drastically decrease their detection rate and we discuss how correct choice of side effects (metrics monitored by HPCs) alongside hardware monitor implementation can help. Our future works include implementing a hardware detection component. We will try to fit a hardware friendly machine learning classifier. Machine learning can help us detect new variants of attacks, plus detect evasive attacks, in which attackers insert nominal operations to fool our mechanism.

References

[1] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. J. Stolfo, "On the feasibility of online malware detection with performance counters," in *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*, A. Mendelson, Ed. ACM, 2013, pp. 559–570.

[2] M. Payer, "Hexpads: a platform to detect stealth attacks," in *International Symposium on Engineering Secure Software and Systems*. Springer, 2016, pp. 138–154.

[3] S.-h. PENG, Q.-f. ZHOU, and J.-l. ZHAO, "Detection of cache-based side channel attack based on performance counters," *DEStech Transactions on Computer Science and Engineering*, no. aiie, 2017.

[4] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 897–912.

[5] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the fourth annual IEEE international workshop on workload characterization. WWC-4*. IEEE, 2001, pp. 3–14.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[7] Y. Yarom and K. Falkner, "Flush+ reload: A high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 719–732.

[8] C. Maurice, C. Neumann, O. Heen, and A. Francillon, "C5: cross-cores cache covert channel," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2015, pp. 46–64.

[9] P. Paul Cristian, "Rsa timing attack," https://github.com/paul110/RSA-Timing-Attack, 2016.

[10] H. Daniel, "web timing attack," https://github.com/dkhonig/web_timing_attack, 2016.

[11] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer. js: A remote software-induced fault attack in javascript," in *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 2016, pp. 300–321.

[12] A. C. De Melo, "The new linuxperftools," in *Slides from Linux Kongress*, vol. 18, 2010, pp. 1–42.