Challenges of Using Performance Counters in Security Against Side-Channel Leakage

Maria Mushtaq, Pascal Benoit, Umer Farooq

Challenges of Using Performance Counters in Security Against Side-Channel Leakage

Maria Mushtaq
LIRMM-CNRS, Univ Montpellier
Montpellier, France
Email: maria.mushtaq@lirmm.fr

Pascal Benoit
LIRMM-CNRS, Univ Montpellier
Montpellier, France
Email: Pascal.Benoit@lirmm.fr

Umer Farooq
Dhofar University
Salalah, Oman
Email: ufarooq@du.edu.om

***Abstract*—Over the past few years, high resolution and stealthy attacks and their variants such as Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown have completely exposed the vulnerabilities in modern computing architectures. Many effective mitigation techniques against such attacks are also being proposed that use system's behavioral parameters at run-time using *Performance Counters* (PCs) coupled with machine learning models. Although PCs, both in hardware and software, have shown promising results when used in the context of security, this paper provides experimental evaluation and analysis of the potential challenges, perils and pitfalls of using these counters in security.**

Keywords–Performance Counters; Side-Channel Attacks (SCAs); Cryptography; Detection; Mitigation; Machine Learning; Security; Privacy.

I. Introduction

One of the biggest challenges in modern computing infrastructure today is that security is not regarded as a system-wide issue and, therefore, preventive measures are vulnerability-specific, limited in scope, and even create new attack surface. To put things in perspective, the two computing legends of modern RISC (Reduced-Instruction Set Computing) architecture, David A. Patterson and John H. Hennessy, stated during their Turing award lecture in 2018 that, "The state of computer security is embarrassing for all of us in the computing field" [1]. The primary reason behind these comments is the fact that, almost everything in modern computing architectures today –from computational optimizations to storage elements and interfaces, from end-user applications to the operating system & hypervisor, and from microarchitecture to underlying hardware –is leading to the discovery of new attack vectors. This is a trend getting further momentum, and worse, a complete attack surface is not known yet. Hardware, which is often considered as an abstract layer that behaves correctly −executing instructions and giving a logically correct output, is leaking critical information as a side effect of software implementation and execution.

Today, computing systems are going through the trough of disillusionment related to the prevailing security. The revelations of security and privacy vulnerabilities in microprocessors, both at software and hardware level, have been appalling. These vulnerabilities affect almost every processor, across virtually every operating system and architecture. We believe that the fundamental reason for existence of these vulnerabilities is that the evolution of computing architecture under Moore's law has been focused almost entirely on the performance enhancement and optimization over the past many decades. To this end, the gains are tremendous as many software and hardware optimization tools and techniques have been proposed to boost performance, such as: hierarchical and shared-memory architectures, pipelining, out-of-order and speculative execution, branch prediction, data/instruction de-duplication, shared libraries, compiler optimizations, use of virtual memory and use of specialized hardware accelerators etc. Security, however, has been often an afterthought all along. But the latest security vulnerabilities, like Spectre and Meltdown along with a large number of sophisticated and stealthy attacks like Flush+Reload, Flush+Flush and Prime+Probe, have demonstrated that security cannot be considered as an afterthought anymore. These vulnerabilities span across multiple levels, from execution units to caches, DRAMs (Dynamic Random Access Memory) and interconnect networks. Thus, security has earned its position as a first-order design constraint today alongside performance, area and power consumption.

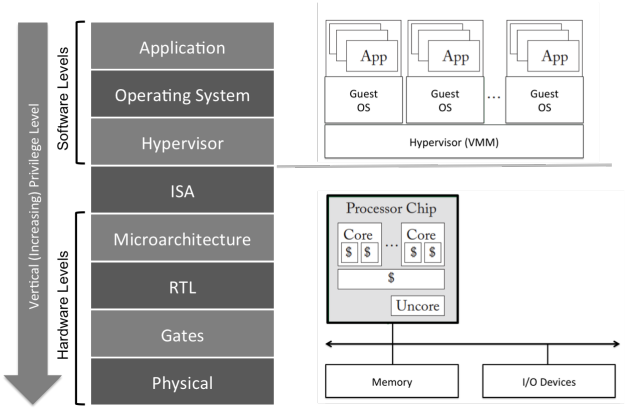

Figure 1. An abstract representation of the full computing stack, covering both software and hardware levels, in a modern general-purpose processor.

Current research in computing architectures is focused on *threat-based design* [2]. For instance, novel architectures like Intel's SGX [3], ARM's TrustZone [4], Bastion [5], AMD's SEV [6] and IBM's 4758 [7] Secure co-processor are a few attempts by major vendors to incorporate security in the design-level specifications. A common characteristic in

these novel architectural designs, however, is that they are incremental designs built on top of general-purpose processor architectures, and expand them with new security features. Thus, they use more or less the same levels of privilege across various layers of computing stack as illustrated in Figure 1.

While it makes perfect sense that a sustainable security can only be achieved by removing the vulnerabilities at the design-time through the proposition of safe software and hardware, researchers are far from achieving that goal. Recent research work suggests that no architecture, new and old alike, is completely immune to information leakage attacks that target all levels (OS, hypervisor, ISA (Instruction Set Architecture), microarchitecture, physical etc.) of computing stacks as shown in Figure 1. In this backdrop, many attempts have been made in recent years to detect, and subsequently mitigate, information leakage attacks using various approaches. The scope of this work, however, is limited to the security solutions against access-driven cache side- and covert-channel attacks. Cache side-Channel attacks are strong cryptanalysis techniques that break cryptographic algorithms by targeting their implementations [8].

Some of the most promising approaches against side-channel information leakage in contemporary architectures to-date are using either hardware and software performance counters or machine learning or a combination of both. Though these approaches have shown promising results in detecting and mitigating a large number of attacks that exploit existing vulnerabilities, the attack vector is still expanding and these approaches need to scale every time a new vulnerability is discovered. Moreover, they rely heavily on the authenticity, determinism, run-time overheads, precision and availability of information that is leveraged through hardware/software performance counters. We believe that, With sophisticated and stealthy attacks appearing everyday, it is just a matter of time that these defenses will not be very effective and the attacker will find a way around such defenses by either manipulating the PCs' information or completely by-passing them undetected.

In this work, we analyze the effectiveness of hardware and software performance counters in security against side-channel attacks. We provide analysis on their benefits, limitations, perils and pitfalls when used to perform detection and mitigation. We validate our analysis with practical results against a large attack vector. The rest of this paper is organized as following. Section II provides related work on the use of HPCsv& SPCs in security. Section III elaborates various PCs and their monitoring tools in existing architectures. Section IV discusses the core limitations of these counters along with experimental data. Section V concludes this paper.

II. Related Work

This section provides the state-of-the-art on various security mechanisms being proposed in recent research work that utilise PCs to perform real-time detection and mitigation of side-channel attacks. Side-channel attack detection techniques are divided into two basic categories; signature-based and anomaly-based detection. Some techniques use a combined approach as well, *i.e.*, signature + anomaly-based detection. Some of the recent research works belonging to the category of *signature-based* detection are reported in [9], [10], [11], [12], [13]. Similarly, in the category of *anomaly-based* detection techniques, many recent research works are reported in [14], [15], [16], [17], [18].

Allaf *et al.* [9] propose a mechanism to inspect Prime+Probe and Flush+Reload attacks targeting AES cryptosystem. Their mechanism uses ML models and HPCs. The proposed mechanism shows good accuracy under isolated conditions, where the attacker and victim are the only load on the system. Another work proposed by Mushtaq *et al.* [10] targets stealthier CSCAs (Cache Side Channel Attacks) like Flush+Flush (F+F). Authors proposed *NIGHTs-WATCH* to detect cache-based side-channel attacks at run-time using ML models coupled with different hardware performance counters that are used to profile victim cryptosystems like RSA (Rivest Shamir and Adleman) and AES (Advanced Encyption Standard) under attack and no-attack scenarios. *NIGHTs-WATCH* being a run-time detection mechanism is evaluated using a variety of metrics like detection accuracy, speed and overhead. Evaluation of the proposed detection technique shows that it can achieve a high detection accuracy with little performance overhead for both attacks even under noisy conditions. Later, this work was extended by Mushtaq *et al.* [12] to include Prime+Probe and other variant attacks under both RSA and AES crypto-systems using the same approach. Their experimental results show consistency for Flush+Flush attack on different implementations of AES as well. Another technique named as *WHISPER* [8] uses multiple machine learning models in an *Ensemble* fashion to detect SCAs at runtime using behavioral data of concurrent processes, that are collected through hardware and software performance counters (HPCs & SPCs). *WHISPER* presents experimental evaluation against Flush+Reload, Flush+Flush, Prime+Probe, Spectre and Meltdown attacks and reports high detection accuracy and low False Positives & False Negatives.

Some of the signature-based detection techniques do not rely on Machine Learning to learn attack signatures. Rather they use thresholds of particular PCs to determine if an attack is in place. One of these works presented in [16], utilizes the values of cache miss rates and page faults of processes to detect an attack. Payer *et al.* in [16] proposed an attack detection framework *HexPADS*, which can detect cache-based side-channel attacks along with Rowhammer [19] and CAIN [20] attacks. *HexPADS* reads status of different performance counters like total executed instructions, total LLC (Last Level Cache) accesses and total LLC misses. It also uses kernel information of processes like total page faults. The proposed detection mechanism basically continuously monitors the cache accesses and misses of all processes. If cache miss rate of a process is found to be higher than 70%, i.e., greater than 70% of cache accesses results into misses, and the same process has a low number of page-faults, the process is reported an attack.

Bazm *et al.* [21] relied on *Intel Cache Monitoring Technology (CMT)* [22] and hardware performance counters and used Gaussian Anomaly detection [23] for detection of cache based side-channel attacks at the level of VMs in IAAS (Infrastructure as a Service) Cloud platforms. Briongos *et al.* [14] proposed *CacheShield* to detect cache side channel attacks on legacy software (victim applications) by monitoring hardware performance events during their execution. The proposed method is implemented at user level and does not require any help from the OS/hypervisor and would be applicable in

cloud environments. As indicated by the authors, this effort is motivated by two main problems of the other detection mechanisms: high detection performance overheads for VMs and requirement of monitoring of both attacker and victim at the same time.

Multiple mitigation techniques have also been proposed against cache-based side-channel attacks in the last decade. These techniques can be categorized into logical & physical isolation techniques, noise-based techniques, scheduler-based techniques and constant time techniques (referring to different cache levels in the cache hierarchy). For instance, logical physical isolation techniques include Cache Coloring [24], CloudRadar [25], STEALTHMEM [26], NewCache [27] and Hardware Partitioning [28]; noise-based techniques include fuzzy times [29], bystander workloads [30]; and scheduler-based techniques include obfuscation [30] and minimum timeslice [31]. Most of these detection and mitigation techniques rely on the low-level behavioral information of the system during execution that is being leveraged for a high-level interpretation and usage through PCs. In Sections III and IV, this paper discusses key benefits and challenges associated with such use of PCs.

III. Performance Counters and Their Use in Security

Performance counters, both software and hardware, have been available in modern processors for more than a decade now with a primary objective to measure the performance of the software when it's being written.

Software Performance Counters (SPCs) are bits of code that monitor, count, or measure events in software, which allow to see patterns from a high-level view. They are registered with the operating system during installation of the software, allowing anyone with the proper permissions to view them. Performance counters can help measure key parts of the software by monitoring the code paths being taken by the software during execution. Like all software, the reliability of SPCs depends on the quality of code and environmental factors. In addition, virtualization in modern computing systems can skew the measurements of processor related counters not because of bad code, but due to how threads are scheduled between the virtual machine, the hypervisor, and the hardware. Almost all operating systems support SPCs such as; page faults, major page faults, minor page faults and invalid page faults. The SPCs are not architecture specific events, but specific to operating systems.

Hardware Performance Counters (HPCs) are special purpose registers built in the microarchitecture of modern processors. HPCs are available as hardware registers that monitor certain events that take place at the CPU level, like the number of cycles and instructions that a program has executed, its associated cache misses and hits, number of accesses to off-chip memory, total number of CPU cycles, number of retired instructions, branch predictions, among several other things. These HPCs are both fixed and programmable in nature. Fixed HPCs are used to measure only specific native events and they cannot measure any other types of event. Programmable HPCs, however, can be programmed to monitor many different events.

Almost all operating systems provide tools to monitor SPCs. There are many high-level libraries and APIs that can be used to configure and read HPCs as well such as: PerfMon [32], OProfile [33], Perf [34], Perftool [35], Intel Vtune Analyzer [36] and PAPI [37] etc. The research work in security reported so far in the state-of-the-art in Section II uses these libraries to access performance counters.

The events that can be tracked with these software and hardware performance counters are usually simple ones, but combined in the right way, they can provide extremely useful insights of a program's behavior and, therefore, constitute a valuable *tool* for run-time analysis. This is the primary motivation behind their recent use in security domain, particularly in the run-time detection and mitigation against various side- and covert-channel attacks that target the execution or the implementation of security-sensitive applications like cryptosystems. Section II details most of such research works.

As an illustrative example of what kind of useful information can be retrieved using performance counters, we consider the case of Prime+Probe cache-based side-channel attack [38], which is a last level cache-based cross-core attack. Like most of such attacks, as shown in Figure 2, this is a three-phase attack in which the attacker fills the cache line(s) with its own content at first, as shown in part (a). This is called the Prime phase. In the second phase, usually a wait phase, the attacker lets the victim program to execute and access whichever memory locations the victim intends to access as shown in part (b). In the third phase, called the Probe phase, attacker accesses the same cache line(s) again, only to find out which particular memory addresses have been touched upon by the victim program. Since there is a significant difference in the amount of time taken to process an instruction if it is supplied to the CPU from cache (i.e., a cache hit) compared to when it is not found in the cache (i.e., a cache miss) and therefore being fetched from main memory.

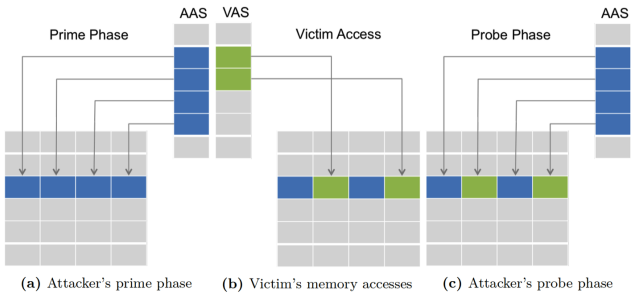

Figure 2. Working principal of Prime+Probe Cache Side-Channel Attack. Here, AAS refers to the Attacker Address Space, whereas VAS refers to the Victim Address Space.

For a computing system under Prime+Probe attack, a careful analysis of the number of total CPU cycles, combined with the cache misses and hits by using performance counters, can reveal a lot of useful information about the system's run-time behavior under attack. Thus, it can be used to subsequently protect the system as well as shown in many recent research works (Section II).

IV. Challenges in using Performance Counters for Security

Although performance counters, both SPCs and HPCs, have shown promising results when used in the context of security, in this paper, we intend to warn the readers/users

about the potential challenges, perils and pitfalls of using these counters in security based on our experiments. Information leakage attacks are becoming stealthy and sophisticated over time. Moreover, they target all layers in computing stack, from logical to physical layers. It is only a matter of time that these attacks will find a way to either *bypass* such detection and mitigation mechanisms that are based on performance counters or find a way to *fool* their measurements. In this section, we provide some experimental evidence and analysis to support this argument. Table I lists the attacks against which we have analyzed the robustness of various performance counters. We have run these attacks alone as well as in various combinations and under variable system load conditions on Intel's x86 architecture to determine whether the information reported by the performance counters is still useful from security perspective. We have experimented with a large set of performance counters with their application scope all layers of computing stack. We then shortlisted only the most relevant counters, as shown in Table II, with respect to the attacks mentioned in Table I for further analysis.

TABLE I. List of Cache-based SCAs that are used as use-cases for the analysis of performance counters on Intel's core i7 machine.

#	Cache SCAs	Attack's Target
1	Flush+Reload	AES & RSA Cryptosystem
2	Flush+Flush	AES & RSA Cryptosystem
3	Prime+Probe	AES & RSA Cryptosystem
4	Spectre	Speculative Execution
5	Meltdown	Out-of-Order Execution

TABLE II. Selected performance counters related to cache-based SCAs mentioned in Table I

Scope of Counter	Performance Counter	Counter ID
L1 Caches	Data Cache Misses	L1-DCM
	Instruction Cache Misses	L1-ICM
	Total Cache Misses	L1-TCM
L2 Caches	Instruction Cache Accesses	L2-ICA
	Instruction Cache Misses	L2-ICM
	Total Cache Accesses	L2-TCA
	Total Cache Misses	L2-TCM
L3-Caches	Instruction Cache Accesses	L3-ICA
	Total Cache Accesses	L3-TCA
	Total Cache Misses	L3-TCM
System-wide	Total CPU Cycles	TOT_CYC
	Branch Miss-Predictions	BR_MSP
	Total Branch Instructions	TBI
	Page Faults	PF

In the following, we provide insights on the discrete challenges that any security mechanism would face while using performance counters. Our analysis is based on extensive experiments that we have performed with a set of PCs being used in existing state-of-the-art security mechanisms and under a large attack vector comprising of most recent side- and covert-channel attacks as shown in Table I.

A. Discernible Information

The PCs allow leveraging a lot of interesting information for high level visualisation and usage in real-time that cannot be observed otherwise. However, our experiments show that their ability to reveal such information gets limited very quickly. For instance, Figures 3–5 illustrate the measured results on the count of L1 data cache misses that were obtained by running various attack (shown in red) and no attack (shown in green) scenarios. Figure 3 shows the L1 data cache misses for P+P attack, while Figure 4 shows the same feature for F+F attack. These two measurements show that, although the PC provides distinguishable information in case of P+P attack, the information is not easily discernible in case of F+F attack due to overlapping behavior despite the same experimental settings. Thus, stand alone, the same PC does not help determining whether a system is under attack or not in case of F+F attack (Figure 4). The situation escalates rather quickly if multiple attacks are running simultaneously, as shown in Figure 5, where we performed experiments with six attacks running in parallel. As illustrated in Figure 5, the information collected on L1 data cache misses is no more discernible and limits the ability of any detection or mitigation mechanism to report an attack scenario based on PC's data alone. Therefore, despite their availability and ability to leverage low-level execution information in real-time, the PCs may not always be helpful in extraction of useful information.

B. Non-deterministic Behavior

Non-determinism is another issue with PCs. Non-determinism refers to a situation where two identical runs of the same program with exactly the same inputs may not produce the same results of monitored events. For applications that are time- and security-critical, determinism is an essential property. However, our experiments show that PCs produce deterministic results only in a strictly controlled environment, which is not always possible to maintain. Deterministic results of PCs also depend on the measuring tools. Authors in [39] report that non-determinism varies the measurements of PCs from $1 - 10\%$. Non-determinism is more an issue of HPCs compared to SPCs. Only few HPCs can produce deterministic results like *retired instruction* when measurements are taken with good tools that can remove sources of contamination from HPCs measurements. Most potentially deterministic events on Intel x86 are affected by the hardware interrupt count [39]. Many important hardware events, such as the ones which measure cache performance and execution cycle counts, are not deterministic on modern out-of-order machines. This severely limits the usefulness of PCs in situations where exact deterministic behavior is necessary. Our experiments show that some of the major sources of non-determinism are linked to the operating system's activities, context switching between concurrently running processes, hardware interrupts, performance overhead of measurements and the precision of the measuring tools. Hardware counters like cache accesses, total execution cycles are non-deterministic on modern out-of-order processors. Therefore, to use HPCs for security applications, one needs to find deterministic HPCs from available counters. In order to avoid false positive and false negatives from the defense mechanisms in security, the sources of contamination must be removed or limited to make the tools reliable.

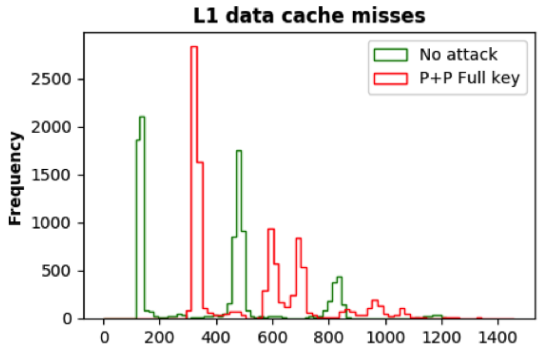

Figure 3. Experimental results measured through performance counters on the effect of P+P attack on L1 data cache misses

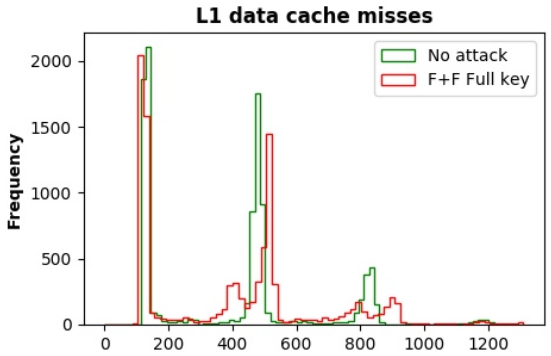

Figure 4. Experimental results measured through performance counters on the effect of F+F attack on L1 data cache misses.

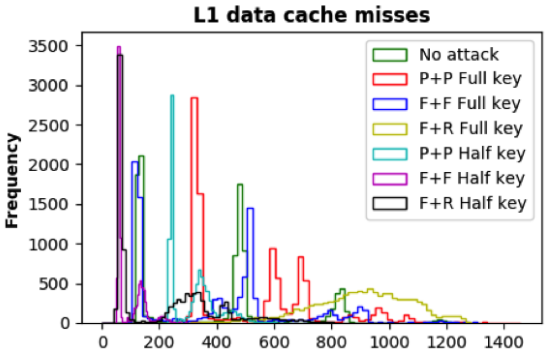

Figure 5. Experimental results measured through performance counters on the effect of multiple attacks running simultaneously on L1 data cache misses.

C. Multiplexing Issues

Although there are hundreds of logical PCs available in modern computing systems to measure various aspects of system's behavior at run-time, physically, there is a limited number of counters that are used to measure and leverage low-level information. Multiplexing allows more counters to be used simultaneously than are physically supported by the hardware. With multiplexing, the physical counters are time-sliced, and the counts are estimated from the measurements. In order to increase the degree of confidence in attack detection and mitigation, many recent techniques use multiplexing of features. In our experiments, we observed that multiplexing can lead to many issues. A naive use of multiplexing could lead to erroneous results in the measurements of PCs that would not be detected by the user. Such errors in measurement occur when the sampling time for these PCs in insufficient to permit the estimated counter values to converge to their expected values. Authors in [40] have also reported similar issues. In such cases, sometimes the PCs do not update their measurement and keep reporting the last sampled/collected data. Another issue with the accuracy of measurements done by multiplexed PCs. Due to lack of sampling granularity under a time-sliced multiplexing model, sometimes the PCs lack accuracy even though they measure and report the events. Such inaccurate or imprecise measurement generate a lot of false positives and negatives by the security mechanisms using multiplexed PCs. Thus, based on our experiments, it is highly recommended to carefully select the minimum number of PCs as features and avoid multiplexing as much as possible. Authors in [40] also mention potential sources of inaccuracy in counter measurements. They point out issues such as the extra instructions and system calls required to access counters, and indirect effects like the pollution of caches due to instrumentation code, but they do not present any experimental data.

D. Performance Overhead

One of the key challenges faced by PC-based security mechanisms is the cost of their sampling/measurement at run-time. The overhead comes from collecting data of PCs during their start and stop and reading of data. The PCs' interface necessarily introduce overhead in the form of extra instructions, including system calls, and the interfaces cause cache pollution that can change the cache and memory behavior of the monitored application. The cost of processing counter overflow interrupts can be a significant source of overhead in sampling-based profiling. A lack of hardware support for precisely identifying an event's address may result in incorrect attribution of events to instruction addresses on modern superscalar, out-of-order processors, thereby making profiling data inaccurate. The performance overhead issue can only be dealt with at the design level of measuring tools in order to keep their run-time overhead and memory footprint as small as possible. Moreover, hardware support for interrupt handling and profiling should be used if possible. Performance overhead can be linked to two distinct usage models of the PCs, namely; counting and sampling. The performance overhead of counting usage model comprises of costs associated with starting and stopping of a PC and reading its values. Whereas, the overhead of sampling usage model comprises of the frequency of sampling or sampling granularity. Though the overhead varies on different platforms and under different measuring tools, it is still a major limitation in the use of PCs in real-time detection and mitigation tools and techniques. Authors in [41] and [42] have reported overheads for various computing architectures.

V. Conclusions

High resolution & stealthy attacks have completely exposed the vulnerabilities in modern computing architectures in recent years. Many effective mitigation techniques against such attacks are being proposed that use Performance Counters coupled with machine learning models. In this work, we analyze the effectiveness of hardware and software performance counters in security against side-channel attacks. We provide analysis on their benefits, limitations, perils and pitfalls when used to perform detection and mitigation. We validate our analysis with practical results against a large attack vector.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development," Turing Lecture, 2018.

[2] Rube B. Lee, Security Basics for Computer Architects, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, September 2013, vol. 8, pp. 1–111.

[3] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for cpu based attestation and sealing," in Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy, vol. 13. ACM New York, NY, USA, 2013.

[4] J. Winter, "Experimenting with ARM TrustZone–or: How I Met Friendly Piece of Trusted Hardware," in 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications. IEEE, 2012, pp. 1161–1166.

[5] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture. IEEE, 2010, pp. 1–12.

[6] D. Kaplan, J. Powell, and T. Woller, "Amd memory encryption," White paper, 2016. [Online]. Available: http://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf

[7] D. L. Osisek, K. M. Jackson, and P. H. Gum, "ESA/390 interpretive-execution architecture, foundation for VM/ESA," IBM Systems Journal, vol. 30, no. 1, 1991, pp. 34–51.

[8] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit, "Whisper: A tool for run-time detection of side-channel attacks," IEEE Access, vol. 8, 2020, pp. 83 871–83 900.

[9] Z. Allaf, M. Adda, and A. Gegov, "A comparison study on Flush+Reload and Prime+Probe attacks on AES using machine learning approaches," UK Workshop on Computational Intelligence, 2017.

[10] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy, ser. HASP '18. New York, NY, USA: ACM, 2018, pp. 1:1–1:8.

[11] M. Sabbagh, Y. Fei, T. Wahl, and A. A. Ding, "SCADET: a side-channel attack detection tool for tracking Prime+ Probe," in 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2018.

[12] M. Mushtaq, A. Akram, M. Bhatti, N. B. R. Rao, V. Lapotre, and G. Gogniat, "Run-time detection of Prime+ Probe side-channel attack on AES encryption algorithm," in Global Information Infrastructure and Networking Symposium, Greece, 2018.

[13] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, M. Yousaf, U. Farooq, V. Lapotre, and G. Gogniat, "Machine Learning For Security: The Case of Side-Channel Attack Detection at Run-time," in 25th IEEE International Conference on Electronics Circuits and Systems, Bordeaux, FRANCE, 2018.

[14] S. Briongos, G. Irazoqui, P. Malagón, and T. Eisenbarth, "CacheShield: Detecting cache attacks through self-observation," in Proceedings of the 8th Conference on Data & Application Security & Privacy. ACM, 2018, pp. 224–235.

[15] Y. Kulah, B. Dincer, C. Yilmaz, and E. Savas, "SpyDetector: An approach for detecting side-channel attacks at runtime," IJIS, 2018.

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

[17] S. Briongos, P. Malagón, J. L. Risco-Martín, and J. M. Moya, "Modeling side-channel cache attacks on aes," in Proc. of the Summer Computer Simulation Conference. Society for Computer Simulation International, 2016, p. 37.

[18] A. Raj and J. Dharanipragada, "Keep the PokerFace on! Thwarting cache side channel attacks by memory bus monitoring and cache obfuscation," Journal of Cloud Computing, vol. 6, no. 1, 2017, p. 28.

[19] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," Black Hat, vol. 15, 2015, p. 71.

[20] A. Barresi, K. Razavi, M. Payer, and T. R. Gross, "{CAIN}: Silently breaking {ASLR} in the cloud," in 9th {USENIX} Workshop on Offensive Technologies ({WOOT} 15), 2015.

[21] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in Fog and Mobile Edge Computing (FMEC), 2018 Third International Conference on. IEEE, 2018, pp. 7–12.

[22] "Benefits of Intel cache monitoring technology in the Intel Xeon processor E5 v3 family," 2018, https://software.intel.com/en-us/blogs/2014/06/18/benefit-of-cache-monitoring.

[23] "Gaussian anomaly detection," 2018, https://wiseodd.github.io/techblog/2016/01/16/gaussian-anomaly-detection/.

[24] M. Godfrey and M. Zulkernine, "Preventing cache-based side-channel attacks in a cloud environment," IEEE Transactions on Cloud Computing, vol. 2, no. 4, Oct 2014, pp. 395–408.

[25] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in International Symposium on Research in Attacks, Intrusions, and Defenses. Springer, 2016.

[26] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-Level Protection Against Cache-Based Side Channel Attacks in the Cloud," in USENIX Security, 2012, pp. 189–204.

[27] F. Liu, H. Wu, K. Mai, and R. B. Lee, "Newcache: secure cache architecture thwarting cache side channel attacks," IEEE Micro Special Issues on Security, vol. 36, 2016.

[28] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," SIGARCH Comput. Archit. News, vol. 35, no. 2, Jun. 2007, pp. 494–505.

[29] W.-M. Hu, "Reducing timing channels with fuzzy time," Journal of computer security, vol. 1, no. 3-4, 1992, pp. 233–254.

[30] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-VM Side Channels and Their Use to Extract Private Keys," in ACM CCS, NY, USA, 2012.

[31] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-VM side-channels," in 23rd USENIX Security Symposium, San Diego, CA, 2014.

[32] PerfMon, "$https://knowledge.ni.com/$," 2018.

[33] OProfile, "$http://oprofile.sourceforge.net/$," 2018.

[34] A. C. De Melo, "The new linux perf tools," in Slides from Linux Kongress, vol. 18, 2010.

[35] P. Tool, "$http://lacasa.uah.edu/$," 2018.

[36] I. V-Tune, "$https://software.intel.com/en-us/vtune-amplifier-cookbook$," 2018.

[37] "Performance application programming interface," in $http://icl.cs.utk.edu/papi/$, 2018.

[38] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in International Conference on Cryptographic Hardware and Embedded Systems (CHES), vol. 9813, 08 2016, pp. 368–388.

[39] V. Weaver and J. Dongarra, "Can hardware performance counters produce expected, deterministic results," in Proc. of the 3rd Workshop on Functionality of Hardware Performance Monitoring, 2010.

[40] J. Dongarra, K. London, S. Moore, P. Mucci, D. Terpstra, H. You, and M. Zhou, "Experiences and lessons learned with a portable interface to hardware performance counters," in Proceedings International Parallel and Distributed Processing Symposium. IEEE, 2003, pp. 6–pp.

[41] S. V. Moore, "A comparison of counting and sampling modes of using performance monitoring hardware," in International Conference on Computational Science. Springer, 2002, pp. 904–912.

[42] M. Maxwell, P. Teller, L. Salayandia, and S. Moore, "Accuracy of performance monitoring hardware," in Proceedings of the Los Alamos Computer Science Institute Symposium (LACSIâ02). Citeseer, 2002.