Special Session: Advances and Throwbacks in Hardware-Assisted Security

Ferdinand Brasser, Lucas Davi, Abhijitt Dhavlle, Tommaso Frassetto, Sai Manoj Pudukotai Dinakarrao, Setareh Rafatirad, Ahmad-Reza Sadeghi, Avesta Sasan, Hossein Sayadi, Shaza Zeitouni, and Houman Homayoun

***Abstract*—Hardware security architectures and primitives are becoming increasingly important in practice providing trust anchors and trusted execution environment to protect modern software systems. Over the past two decades we have witnessed various hardware security solutions and trends from Trusted Platform Modules (TPM), performance counters for security, ARM's TrustZone, and Physically Unclonable Functions (PUFs), to very recent advances such as Intel's Software Guard Extension (SGX). Unfortunately, these solutions are rarely used by third party developers, make strong trust assumptions (including in manufacturers), are too expensive for small constrained devices, do not easily scale, or suffer from information leakage. Academic research has proposed a variety of solutions, in hardware security architectures, these advancements are rarely deployed in practice.**

I. Introduction

Hardware-assisted security promises to solve many long-existing problems of vulnerable software. Hardware security features are used to store and protect sensitive state such as cryptographic keys, to isolate a minimal set of security critical software, or to implement security critical functions directly in hardware. The assumptions are that hardware is less likely to have vulnerabilities, and by minimizing the security critical software its complexity and thus the likelihood for vulnerabilities is reduced. Hardware-assistance is also used to support software security solutions, like control-flow integrity (CFI) [1], e.g., to improve the performance [2].

The development, standardization and deployment of hardware security primitives has mainly been impelled by industry. Academia has mostly been focused on utilizing those hardware primitives, often in new and creative ways, to build security mechanisms and architectures for various scenarios, ranging from small embedded systems to cloud computing.

For instance, security solutions for mobile devices [3] have mostly been based on ARM TrustZone [4], which is the Trusted Execution Environment (TEE) implementation widely deploy in these primarily ARM-processor based devices.

The security and privacy concerns of users and businesses with regard to cloud computing have been the focus of research for many years. Early solutions relied on Trusted Platform Modules (TPMs) and software isolation based on trusted hypervisors [5]. With the introduction of Intel's Software Guard Extensions (SGX) [6]–[8] many solutions have been published aiming to secure cloud applications using TEEs [9]–[12].

Another research direction with the same goal investigates purely cryptographic solutions, i.e., homomorphic encryption (HE) or secure multi-party computation (SMPC). However, these solutions are highly impractical due to their massive overhead in computation time and communication costs.

TEEs have been proposed as a substitution for cryptography-based secure multi-party computation been proposed in [13]–[15]. Ohrimenko et al. [16] adapts several machine learning (ML) algorithms, including neural networks, to prevent cache-based side-channel attacks in scenarios where multiple institutions use Intel SGX to securely share their datasets for training and evaluation of joint ML models. In [17], the authors introduce a similar protection mechanism that is efficient enough for real-time data processing: instead of preventing memory accesses that depend on sensitive data, they add noise to memory traces by accessing dummy data. The very recent Chiron [18] system allows a user to train a model using the computing resources of a cloud service provider while the training data remains hidden and the resulting model can only be accessed as a black box. VoiceGuard [19] allows secure and privacy-preserving speech recognition in public cloud.

However, while hardware-security mechanisms can help to improve the security of systems it is not a magic bullet. Many problems remain even when using them, for instance, they are often vulnerable to hardware and hardware-related attacks like side-channel attacks. Also, the security critical functions, despite being isolated through hardware security architectures, remain vulnerable to runtime attacks. Even worse, hardware-security mechanisms give rise to additional problems, for instance, they require strong trust in manufacturer of hardware components, or they often provide limited access to third-party developers. Hardware-security mechanisms are usually integrated into legacy system, as an afterthought. As a consequence they are not always scalable and cannot provide comprehensive security guarantees.

In this paper, we discuss different hardware-security architectures and primitives, with their limitations and possible attacks. We start with physically unclonable functions (PUFs), followed by co-processor based approaches. Afterwards, we discuss TEEs, starting with ARM TrustZone. The focus will be on Intel's recent SGX, for which we detail on different attack vectors. Looking closer at side channels, we will elaborate on various defense strategies and discuss the challenges in

F. Brasser, T. Frassetto, A. Sadeghi, and S. Zeitouni are with Technische Universität Darmstadt, Germany. Email: {ferdinand.brasser,tommaso.frassetto,ahmad.sadeghi,shaza.zeitouni}@trust.tu-darmstadt.de

A. Dhavlle, S. M. Pudukotai Dinakarrao, S. Rafatirad, A. Sasan, H. Sayadi, H. Homayoun are with George Mason University, United States. E-mail: {adhavlle,spudukot,srafatir,asasan,hsayadi,hhomayou}@gmu.edu

L. Davi is with University of Duisburg-Essen, Germany. E-mail: lucas.davi@uni-due.de

defeating side-channel attacks in the context of SGX.

In addition to the aforementioned challenges, recent studies have demonstrated that malicious activities on the hardware-level ranging from application-based malware to side-channel attacks can be effectively recognized by classifying anomalies in the low-level feature spaces such as microarchitectural events collected by Hardware Performance Counter registers. The last section of this paper devotes to the comprehensive analysis of defense mechanisms against malicious applications and side-channel attacks targeting the available on-chip hardware performance counters.

II. Physically Unclonable Functions (PUFs)

Since their celebrated invention, PUFs have been introduced as central building blocks in security architectures and cryptographic protocols. A PUF is a noisy function embedded in a physical object that generates an object-specific output (*response*) to a given input (*challenge*). Several attempts to model the most relevant security properties of PUFs have been made [20], [21]. Hence, PUFs are assumed to fulfill the following properties: physical unclonability (infeasible to create two identical PUF instances), unpredictability (infeasible to predict responses of unknown challenges), robustness (with high probability, a PUF instance generates the same response for the same challenge) and tamper-evidence. Among the common deployment scenarios of PUFs are secure cryptographic key generation, device authentication/identification and binding software to hardware. For these purposes, a variety of PUF constructions have been proposed. The most intuitive ones for the integration into electronic circuits are the silicon PUFs, which fall into different categories including *Delay-based* PUFs and *Memory-based* PUFs. Delay-based PUFs exploit race conditions in integrated circuits (e.g., Arbiter PUFs and Ring Oscillator PUFs). Memory-based PUFs exploit the process variation in volatile memory elements e.g., dynamic random access memory (DRAM), static RAM (SRAM), flip-flops and latches.

Along the lines, PUFs have been subject to various evaluations as well as attacks. The attacks range from non-invasive modeling to (invasive) physical attacks or a mix of both, referred to as hybrid attacks. Modeling attacks usually utilize ML techniques, which exploit the linear structure of a PUF to derive numerical models. During the last decade, several new or modified PUF constructions were introduced as modeling-resilient PUFs, however, later on they have been attacked using advanced ML techniques [22], [23]. On the other hand, although PUFs have been considered to be tamper-evident, it has been shown that they are vulnerable to all kinds of physical attacks including side-channel, (semi-) invasive and fault injection attacks without affecting their functionality [24], [25]. Semi-invasive attacks from the IC backside of a PUF chip have been demonstrated to be feasible, as well [26]. Moreover, hybrid attacks on PUFs combining physical attacks and ML techniques for PUF modeling have been proven to be effective [27]–[29].

In the opposite direction, several new PUF constructions have been proposed. MEMS PUFs [30] leverage Micro-Electro-Mechanical Systems (MEMS)-based sensors as a source of entropy to generate a secret key with the help of a fuzzy extractor. However, it requires at least 23 sensors to generate a strong 128-bit key. Other PUFs that are based on non-linear electronic characteristics, e.g., the voltage-transfer PUFs [31] and the current-mirrors PUFs [32], were introduced as ML-resilient PUFs. These PUFs have been proven to be vulnerable to more advanced ML techniques [33]. Besides that, several PUFs based on emerging nano-technologies have been proposed [34]. Other PUF proposals that leverage emerging non-volatile memory technologies include Memristor/Resistive Random Access Memory (RRAM) [35] and Magnetoresistive Random Access Memory (MRAM) and its advanced version Spin-Transfer Torque MRAM (STT-MRAM) [36].

A. Conclusion

While current PUF constructions have been proven to be vulnerable to physical and modeling attacks, recent PUF constructions that are based on emerging technologies lack a comprehensive security evaluation framework, which is very crucial to gain confidence that such PUF constructions can be deployed in real-world scenarios. Therefore, it is obvious that the construction of PUFs that are resilient to advanced modeling attacks as well as physical attacks is still an open challenge and a prerequisite for the security of PUF-based protocols.

III. Co-Processor Solutions

One of the most widely available co-processor based hardware-assisted security solutions is the TPM [37], which extends commodity computers with a hardware root of trust for a number of security services, including *device identification*, *remote attestation*, and *secure storage* and *sealing*. These capabilities allow the protection of data at rest and to ensure the initial integrity of a computer's software, i.e., ensure that software like the operating system was not modified.

In later version of TPM and in conjunction with the CPU, the concept of dynamic root of trust for measurement was introduced (Intel Trusted Execution Technology [38] and AMD Secure Virtual Machines [39]). It allows to reset the computer at runtime to a secure state, from where the integrity of all loaded software can be checked. Flicker [40] has demonstrated that this mechanism can also be used to build a TEE like mechanism.

However, since the TPM only ensures the integrity of software at load time TPM-based solutions do not provide protection against runtime attacks like Return-oriented Programming (ROP) [41]–[46].

A. Conclusion.

Ensuring the security and trustworthiness of entire platform is hard (or even impossible) to achieve due to the large software stack on today's platforms. Therefore, protection mechanisms are needed that allow the fine-grained isolation of software components. Additionally, the integrity of such components must be ensured both, at load-time and run-time.

IV. ARM TrustZone

TrustZone represents a set of security enhancements to ARM's processor designs and Systems-on-Chip. TrustZone enhances the processor, memory (including caches), and peripherals. A TrustZone-enabled processor can execute instructions in two security modes at any given time. The secure and normal world both manage their own address spaces using the traditional privilege levels for separation of the OS kernel and application code.

The processor can switch from normal to secure world via an dedicated secure monitor call (`smc`) instruction. When an `smc` instruction is invoked from normal world, the processor performs a context switch to the secure world (via the monitor mode) and freezes the execution of the normal world.

TrustZone can separate physical memory into two partitions, with one partition being exclusively accessible by the secure world. This isolation is enforced by the memory controller (TZASC). While the normal world cannot access memory assigned to secure world, the secure world can access normal world memory.

A device running ARM TrustZone boots up in the secure world. After the secure world finished its initial setup, it switches to the normal world and boots the legacy operating system. Most TrustZone-enabled devices are configured to support *secure boot*, i.e., the boot loader cryptographically checks the secure world OS prior to execution [47]. In fact, many vendors lock their devices against end-user modification via secure boot, to ensure the integrity of the secure world. This allows them to treat the secure world as part of their Trusted Computing Base (TCB).

A. TrustZone Attacks and Limitations

Despite TrustZone's implementation and wide-spread deployment, the underlying TEE is mainly used by the vendors for their own purposes, and hence a flourishing landscape of secure mobile services is largely missing, even more than a decade after TrustZone was initially released [48].

In practice, bugs in TrustZone-enabled applications expose a large number of devices to real-world security threats, as continuously demonstrated by security researchers across device families and hardware vendors: TrustZone's isolation has been repetitively broken [49]–[56]. Google's ProjectZero [57] recently summarized the main flaws of the current design of TrustZone as follows: it combines (i) weak isolation between trusted applications (TAs) in TEE, with (ii) TCB expansion, and (iii) highly privileged access to the platform, making TrustZone a high-value target for attackers. These problems strongly limit the use of TrustZone for security-critical applications which will benefit both developers and users.

B. Conclusion

The fact that TrustZone supports only a *single* isolated execution environment that has to be shared by all trusted applications leads to weak isolation among them, which in turn is co-responsible for the restrictive access policies of the device manufacturers. To enable an open and secure multi-stakeholder system multiple, mutually strongly isolated execution environments are necessary.

V. Intel Software Guard Extensions

Intel Software Guard Extensions (SGX) enables processing of confidential data on untrusted systems [6]–[8]. SGX introduces the concept of *enclaves*, which are programs executed in isolation from *all* other software on a system, including privileged software, like the operating system (OS) or a hypervisor.

Enclaves are loaded as part of a host process and are embedded in its virtual memory, like a library. The initial content of an enclave is loaded from unprotected memory, hence, it can be manipulated and is not kept confidential. Therefore, confidential data must be provisioned to an enclave over a secure channel *after* it has been created. To ensure that secret data is not sent to a malicious (or maliciously modified) enclave, the integrity and authenticity of an enclave needs to be verified before provisioning secret data. To enable this, SGX provides a security service called remote attestation (RA).

Once available inside an enclave, secret data can be encrypted using an enclave-specific key and written to untrusted storage, e.g., the hard disk. This *sealing* mechanism allows an enclave to use secret data across multiple instantiations.

A. Attacks on Intel SGX

SGX protects enclaves against *direct* accesses, however, code *inside* the enclave can still be attacked through runtime attacks. Also effects of an enclave's computations can be observed through side channels allowing to deduce sensitive information. Subsequently we discuss each of these attack vectors in detail.

RunTime Attacks. In the ideal scenario, the enclave code only includes minimal carefully-inspected code, which could be formally proven to be free of vulnerabilities. However, legacy applications can be adapted as well to run inside SGX enclaves with relatively minor modifications. Formally verifying or manually inspecting such complex legacy software is not feasible, meaning that the same *memory-corruption vulnerabilities* that plague legacy software are also very likely to occur in those complex enclaves. Such vulnerabilities allow an attacker to mount a *runtime attack* to induce unauthorized program actions.

Only recently, Kuvaiskii et al. presented SGXBounds [58] that offers protection against out-of-bounds memory accesses. Lee et al. [59] presented the first memory-corruption attack against SGX. Their attack, called Dark-ROP, is based on several oracles and return-oriented programming (ROP) [41]. The oracles inform the attacker about the internal status of the enclave execution, whereas ROP maliciously re-uses benign code snippets (called *gadgets*) to undermine non-executable memory protection. Dark-ROP is based on principles of blind ROP [60]: if an application is not randomized, or it is not re-randomized after crashing, crashes leak useful information to the attacker. This allows Dark-ROP to extract secret code and data, as well as undermine remote attestation. However, Dark-ROP requires a constant, non-randomized memory layout as the oracles frequently crash enclaves.

Hence, to address the Dark-ROP attack, Seo et al. [61] demonstrated an implementation of SGX randomization called SGX-Shield. Randomization schemes such as SGX-Shield [61] challenge the assumption of a constant memory layout, since the memory layout changes every time the enclave is constructed. Further, SGX-Shield makes traditional exploitation techniques significantly hard to apply because it employs fine-grained randomization and non-readable code.

Biondo et al. [62] propose code-reuse attacks against enclaves built on top of the Intel SGX SDK that also undermine randomization techniques such as SGX-Shield. By abusing preexisting SDK mechanisms, their attacks provide full control of the CPU's general-purpose registers to an attacker able to exploit a memory corruption vulnerability. Controlling registers is essential in any code-reuse attack. For instance, they can prepare data for subsequent gadgets or set arguments for function calls. To this end, two new exploitation primitives are developed: the ORET and CONT primitive attack technique.

The ORET primitive is based on abusing the function `asm_oret` from the tRTS library in the Intel SGX SDK. Normally, this function is used to restore the CPU context after an OCALL. However, when exploiting this function by means of a code-reuse attack, the ORET primitive gives control of a subset of CPU registers, including the register that holds the first function argument and the instruction pointer. In contrast, the CONT primitive abuses the function `continue_execution` from the tRTS, which is meant to restore the CPU context after an exception. This primitive requires the ability to call that function which is achievable by exploiting a memory corruption vulnerability affecting a function pointer. This primitive yields full control over all general-purpose CPU registers. In addition, this attack primitive can be combined with the ORET primitive to also apply it to controlled stack situations.

In preparation for the exploit, the attacker performs static analysis on the enclave binary to determine the gadgets she wants to reuse. In particular, the attacker starts by determining the offsets of `asm_oret` and `continue_execution`. Since they are part of the loader, which is challenging to randomize by existing randomization solutions such as SGX-Shield, those offsets will not change at runtime. Next, the attacker constructs a gadget chain consisting of a sequence of gadgets which will perform the desired malicious activity, and defines the register state that should be set before executing each gadget. The primitives work by abusing functions intended to restore CPU contexts by tricking them into restoring fake contexts, thus gaining control of the registers. In contrast to a standard ROP exploit, which is usually self-contained, the attacks require a number of auxiliary memory structures to hold these fake contexts and execute the primitives. This setup allows triggering the first CONT primitive to start an ORET+CONT loop. Every cycle will execute a gadget and advance the chain, thus running the attacker's payload. Specifically, the proof-of-concept attack extracts cryptographic keys used during the remote attestation process. Once an attacker is in possession of those keys, she can impersonate the enclave when communicating with the remote server. These attacks apply to any enclave developed with the Linux or Windows Intel SDK for SGX.

As discussed, building randomization-based defenses for SGX enclaves is challenging as it requires careful support of SDK library code and additional protection of SGX context data to mitigate the threat of runtime attacks against SGX. On the other hand, enforcement-based defense techniques against runtime attacks such as control-flow integrity [1], [63] could be leveraged for SGX enclave code. This would ensure that the enclave's program flow always adhere to a pre-defined control-flow graph.

Side-channel Attacks. Side-channel attacks on software in general, and SGX in particular, come in many different forms. Any kind of resource use that is influenced by the software's execution and can be observed by the adversary can serve as a side channel. For instance, the use of electricity as well as effects thereof like electro-magnetic emission, or the use of shared CPU caches. In this context we focus on *software* side channels, i.e., such that are observable by a software program running on the target machine, precluding physical or hardware side-channel attacks.

In the realm of software side-channel attacks a number of distinct variants exist. On one hand, different shared resources can be used as a side channel, like the different caches of the CPU, or the virtual memory management. On the other hand, side-channel attacks can target different information, including sensitive access patterns to data as well as secret dependent code execution paths.

Controlled channel attack. Xu et al. [64] demonstrated page-fault side-channel attacks on SGX, where an untrusted operating system exfiltrates secrets from enclaves by tracking memory accesses at the granularity of memory pages.

Cache side-channel attacks. Lee et al. [65] use branch shadowing to infer the control flow of an enclave. Their approach requires the victim enclave to be interrupted at a high frequency, which enables effective detection methods [66], [67].

Schwarz et al. [68] study a scenario, where an unprivileged attacker process (hiding in an enclave) is spying on the L3-cache utilization of another process (or enclave).

CacheZoom [69] attacks an AES implementation through L1 cache by interrupting the victim, and thus increasing the temporal resolution of the attack. Enclave exits introduce noise in a subset of cache lines rendering them unobservable. Additionally, the interrupts make the attack easily detectable [66], [67].

Götzfried et al. [70] also attack AES on L1. They run the victim uninterrupted to avoid disturbance due to enclave exits. However, their attack assumes synchronization (collaboration) between the victim and the attacker – an assumption which typically does not hold in practice.

Brasser et al. [71] show that side-channel attacks are not only dangerous for cryptographic algorithm but can be used to extract sensitive information from a much broader range of data processing algorithms. The authors demonstrate this by extracting genomic data from an SGX enclave running a genome analysis algorithm. Unlike previous works their attack does not requires interrupts or makes synchrony assumptions, which makes it harder to detect and easier to deploy in practice.

Recent works [72], [73] have investigated the possibility of leaking information through a side-channel with granularity smaller than a single cache line.

CacheBleed [72] exploits cache *bank conflicts* to leak fine-grained information. However, this attack does not apply to SGX CPUs due to an updated cache design.

MemJam [73] uses read-after-write false dependencies to introduce latency when a victim program reads data with a specific page offset. By measuring the run time of the victim program a high number of times while *jamming* different page offsets, the attacker can infer which offsets are read more often by the victim. This attack can leak information with a four byte granularity, but requires an extremely high number of runs (*50 million runs* for an attack against a simple and deterministic SGX enclave).

B. Side-channel Defenses

New cache architectures. Cache-based side-channel attacks can be addressed by changes in the cache architecture. The two common approaches are (i) cache partitioning [74]–[77], where the cache is divided into partitions that are not shared between processes, and (ii) cache access obfuscation [76], [78]–[80], where the goal is to obfuscate the side-channel information obtained by the attacker. Such defenses require hardware changes and are limited to cache attacks. Such approaches do not defend against other side-channels, e.g., based on page-faults.

Transactional memory. Some of the known SGX side-channel attacks interrupt the victim enclave repeatedly [64]. Corresponding defenses enable the victim enclave to detect interruption and take counteractive measures, such as stopping its execution. T-SGX [66] leverages the Intel Transactional Synchronization Extension (TSX) to detect asynchronous enclave exits, e.g., due to interrupts from page faults. Déjá Vu [67] monitors the execution time of an enclave to detect a slowdown due to frequent interrupts. These defenses do not prevent attacks that work interrupt-less [68], [70], [71].

Cloak [81] uses TSX to preform atomic memory operations that hide sensitive memory accesses. Before sensitive memory is accessed, all cache lines are touched (primed) by the enclave, and thus the adversary learns nothing about the enclave's sensitive accesses. Cloak relies on the developer to annotate sensitive data structures that should be protected from side-channel attacks and requires the TSX feature that is not supported by all SGX processors.

ORAM and oblivious execution. Oblivious RAM (ORAM) [82]–[87] refers to schemes that hide the memory access pattern of a trusted client (e.g., CPU or network client) to an untrusted and encrypted memory (e.g., DRAM or storage server) by introducing fake accesses and shuffling the encrypted memory elements such that the observable access pattern is independent of the actual access pattern. ORAM systems are typically designed for a model where the trusted client has internal secure memory for maintaining required meta-data. If ORAM is used to hide all memory accesses of an enclave, the client would be the enclave and the RAM would to be considered the untrusted memory. Since an adversary can observe the enclave's memory access patterns, the enclave needs to access also the internal meta-data in an oblivious manner which increases performance overhead.

Oblivious execution architectures [88]–[90] attempt to hide all observable effects of program execution, including both memory accesses (code and data) and timing information. Oblivious execution on standard processor architectures is extremely expensive, and thus oblivious execution systems leverage custom hardware.

Data randomization for SGX. Raccoon [91] provides oblivious data access only for developer-annotated enclave data, thus reducing the overhead. Memory accesses are hidden by either ORAM or streaming over entire data structures.

DR.SGX [92] offers side-channel defense with an adjustable trade-off between security and performance. In DR.SGX the entire data memory of the application is randomized on a cache-line granularity: like in ORAM schemes, the location of each block is moved to a randomized location. In contrast with ORAM, which requires big data tables to keep track of the location of each block, DR.SGX uses a pseudo-random permutation function which only requires minimal secret data: as a result, while ORAM schemes need to implement further costly protections to access the metadata without leaking information to the adversary, DR.SGX can access its secret data easily and efficiently. The permutation function is derived from small-domain format-preserving encryption techniques and leverages the fast hardware implementation of AES on Intel processors. Additionally, DR.SGX can re-randomize the data at a configurable interval, giving developers the possibility to prioritize performance or security.

ZeroTrace [93] is an oblivious data structure framework for SGX that runs on top of a software memory controller. ZeroTrace is designed to hide memory access to resources *outside* of an enclave, e.g., to the hard disk drive. Importantly, it is not designed to make *all* memory accesses of an enclave to its own main memory oblivious. Furthermore, ZeroTrace requires the developer to use the memory controller interface for all access that should be protected.

HardIDX [94] provides oblivious database accesses from an SGX enclave to external storage.

C. Emerging Hardware-based Attacks

Attacks like Meltdown [95], Spectre [96] and Rowhammer [97]–[99] have impressively demonstrated that the complexity of our modern computer systems bears new threats. Meltdown and Spectre operate on a level below the access control enforces by SGX and can therefore undermine its security guarantees [100].

D. Conclusion

While SGX's isolation adds security for many usage scenarios it cannot guarantee comprehensive security under all circumstances. In fact, the remaining attack vectors, as discussed above, must be considered when using SGX. It is the developers and users responsibility to harden his code against runtime attacks as well as side-channel attacks. Also, security solutions need to be reevaluated whenever new attacks are discovered.

VI. Exploiting Performance Counters for Hardware Security

The security of a system can be compromised by either side-channel attacks or by executing malicious applications infecting the system. In addition to the aforementioned defenses, here we present the detection mechanisms and measures that take advantage of built-in hardware components i.e., hardware performance counters (HPCs) to capture the running application behavior for security. HPCs are special purpose registers embedded inside modern microprocessors to monitor and capture different microarchitectural events. The primary purpose of HPC is to analyze and tune the architectural level performance of running applications [101]–[104]. Recent works have proposed to utilize the HPCs for securing the hardware systems against both malware (application execution based attacks) and side-channel attacks. We present the analysis in-detail below.

A. HPCs for Malware Detection

Malware is a piece of code written by the attacker to perform intended malicious activities such as information leakage, data stealing, and gaining unauthorized access without the consent of the user. Researchers have suggested both the signature-based and anomaly detection based malware detectors using the HPCs. In this section, we discuss the latest efforts on hardware-assisted malware detection.

The work in [106] is one of the first works to study the suitability of HPC data for detecting the malware. It uses ML models for malware detection. The primary focus of this work is on detecting malware in mobile OS such as Android. In addition, it has demonstrated the capability of employing HPC information for detecting malware such as Linux rootkits, and cache side-channel attacks on Intel and ARM processors. In [107], HPCMalHunter, which is a behavioral online malware detector that predicts the existence of malware with high accuracy by deploying support vector machine (SVM) is proposed. As the number of available microarchitectural events are large and the number of HPCs that can be accessed simultaneously are small, a Singular Value Decomposition (SVD) based feature reduction is deployed for selecting the prominent microarchitectural events. Thus, only prominent microarchitectural events are monitored through HPCs.

Similarly, detection of malware specifically for Kernel-level rootkit attacks with the aid of HPCs is proposed in [108]. In this work, ML classifiers are trained using the HPCs collected from benign and rootkit applications for detection and classification. It has been found that the rootkits employing direct kernel object manipulation (DKOM) do not significantly impact the HPCs, which makes it hard to detect with the aid of simple HPCs. The works such as Numchecker [109] employ HPCs' information for detecting the rootkits. Numchecker is a virtualization based framework devised to detect malicious modifications of the guest VM's system calls using the behavior of hardware events. The work in [110] proposes dynamic integrity checking of programs during runtime for malicious activities using HPC information. In this work, a comparison of the HPC values is used for malware detection. To address the associated memory overheads with storing HPC patterns for malware detection, a "sample-locally-analyze-remotely" technique is proposed in [111].

A malware-aware processor (MAP) is proposed in [112], [113], where different sub-semantic features of low-level microarchitectural events are explored for malware detection, such as: (1) features based on executed instructions; (2) features based on the memory address patterns; (3) features based on architectural events. However, MAP suggests changes in the microprocessor pipeline for evaluating sub-semantic features and detecting malware in real-time.

To facilitate runtime malware detection, [114], [115] proposes leveraging limited available number of HPCs in which the microarchitectural events are chosen based on the systematic feature reduction. It has been found that different ML classifiers achieve different performance across different malware classes. Furthermore, the accuracy drops with the number of HPCs used for detecting the malware. However, to enhance the performance, the work in [114]–[117] proposes use of ensemble ML-based solutions for effective runtime malware detection using low-level microarchitectural features. For feature reduction, [114], [115] employ a systematic approach to select the top events from the entire set of available events by using Correlation Attribute Evaluation technique.

Tang et al. [118] deployed unsupervised learning that employs low-level features (HPCs) for detecting ROP and buffer overflow attacks by detecting the anomaly patterns in the HPC patterns. It uses samples from HPCs to train unsupervised ML techniques for detecting deviations in program behavior that occurs due to a potential malicious attack. Although unsupervised algorithms can be more effective in detecting new malware and attacker evolution, they are complex in nature requiring more complex hardware implementations.

One of the recent works [119] performs the cross validation of HPC based malware detection. The results have shown a larger variation in performance across when using different ML classifiers. Similar variations are observed when employing different ML classifiers in the previous works as well. Additionally, [119] presents an adversarial malware sample where a notepad++ application is fused with ransomware application that encrypts the data leading to result in same HPC traces as a benign application. This poses the challenge that for sophisticatedly crafted malware samples, simple anomaly-detection based detection techniques are might not be sufficient, and thus needs to be improved and used in conjunction with other malware detection techniques.

In addition to malware detection that only alters the application control-flow, use of HPCs have also shown to be effective for malware targeting firmware modifications. ConFirm [120] proposes a lightweight technique for detecting malicious modifications in firmware libraries. This work performs a comparison of low-level hardware events (HPCs) for detecting malicious activities. Similarly, for firmwares involving complex control flows, a ML-based classifier using HPC information is proposed in [121].

B. *HPCs for Side-channel Attack*

HPCs are also employed for detecting side-channel attacks. As mentioned, side-channel attacks target secret keys or information used in cryptographic [105] and secure operations. The attack detection using HPCs are presented below.

The proposed works [122], [123] can detect Flush+Reload, Prime+Probe and Flush+Flush. The Flush+Flush is relatively robust as it uses only Flush instruction and hence can bypass defenses based on timing as a parameter. The CloudRadar presented in [122] employs two distinct steps for detecting the side-channel attacks: Signature-based detection and anomaly-based detection. The former one works by comparing monitored application with pre-defined attack signatures and the latter one works by detecting the deviation in the application's behavior by comparing it with its normal behavior. However, the CloudRadar requires an entire physical CPU for monitoring other VMs and applications might be an overhead and adds up to the price that the cloud service users have to pay. To overcome the limitations, CacheShield is proposed in [123].

CacheShield [123] aims at reducing the overhead caused when all the VMs in a Cloud are to be monitored continuously for security threats. It is kind of an on-demand solution where a user (VM) notifies the CacheShield which in turn utilizes the HPC info from the PMU to decide whether the user/VM is under attack or not. CacheShield proposes change-point detection technique for detecting abrupt changes in the distribution. In contrast to traditional ML-based anomaly detection techniques, change-point detection method is self-learning. The CacheShield works based on the assumption that the performance of protected algorithms are affected under attack situations and the detection method in itself has minimum impact on the system performance. In the event of an attack, it can then implement one of the numerous available mitigation strategies (such as adding noise by frequently flushing cache lines [123]; hiding the true secrets in a process by generating dummy ones [123]), and less protected implementations in case of no attacks with low overheads. The above mentioned defenses protect LLC from the aforementioned attacks.

Detection of Flush+Reload side-channel attack that exploits the dependencies of HPC data and ML is proposed in [124]. In the first method, detection is done by obtaining the correlation between victim and spy process by analyzing the data obtained from Quickhpc tool [125], which is similar to `perf_stat`. The correlation of total L3 cache accesses over time is seen as a good indicator to differentiate the victim from the spy process. In the second method that employs ML techniques (such as neural networks), though computationally intensive but has proven to provide better results and do not require the data to be pre-processed. In addition to the above methods, another detection mechanism is proposed in [124], where the data samples obtained from the spy processes are considered as "normal" and from any other processes as "anomalies". The advantage of this method is that it requires less time and data to model each spy, as the number of spy programs available are limited. However, for unseen spy applications, the classifier has to be trained again. As such, this technique is more suitable for defending against known attacks.

The work in [126] is one of the first side-channel attack detection methods to leverage Intel's CMT (Cache Monitoring Technology). This provides two advantages: improves system performance by observing the cache occupancy of the VM and then applying resource usage limitation through CAT (Cache Allocation Technology), thereby reduces the shared resource contention and mitigates cache side-channel attacks. During an attack, as the attacker VM tries to evict cache lines multiple times from the LLC, this aggressive behavior of the VM is detected by the CMT and thus the side-channel attack is detected. This technique is effective to attacks like Prime+Probe, Flush+Reload, Flush+Flush and Evict+Time.

The work in [127] focuses on the temporal scheduling of the processes to reduce hardware/functional units contention and information sharing, eventually leading to minimizing the risk of side-channel attacks. The [127] suggests to devise a scheduler that can peek into an application/process and determine what kind of operation it would perform during its runtime based on the modeled ML predictors and thus schedule them such that no two such applications are scheduled on the same core to avoid hardware unit contention and improve performance. In addition to scheduling, HPCs are also monitored to recognize memory intensive applications and schedule them on separate cores to avoid information sharing between two or more applications. Thus, the information leakage through side-channels can be reduced.

In addition to these attacks, the Spectre [96] and Meltdown [95] are some of the recently introduced attacks that have seen to be one of the most devastating side-channel attacks discovered so far. The work in [128] proposes use of HPCs for detecting such advanced attacks. As we know that Spectre [96] and Meltdown [95] leave footprints due to 'page fault'. Hence, these can be defended and detected by exploiting the same. These defenses capture the "segfaults" inside the operating system. The approach is successfully tested by the kprobe tool [129], as any application generating too many segfault errors is scarce. This defense mechanism is built based on the assumption that the attacker does not employ Intel's TSX, as the `SIGSEV` (segfault) error will not be thrown and the defense mechanism would fail.

Other robust defense technique to detect and mitigate the Meltdown and Spectre attacks using HPCs is presented in [128]. This work employs HPCs to detect and mitigate the Spectre and Meltdown attacks by using `LLC_References`, `LLC_Misses`, `LLC_load_misses`, and `LLC_loads` counters. From the experiments in [128], it has been found that since these attacks target LLC and during the attack there are more cache loads/misses due to the fact that the attack flushes the cache content, the `LLC_References` and `LLC_Misses` varies tremendously compared to normal programs. Thus, based on these counters and by calculating the miss Rate, Spectre and Meltdown attacks can be detected.

The side-channel attacks, Spectre, and Meltdown exploit the vulnerabilities by measuring the temporal difference between the cache access (which varies if data is hit, miss, and so on, followed by utilizing this timing differences to identify executable, non-executable, mapped and unmapped pages in the memory). Modern state-of-the-art attacks [130] use Intel's TSX to measure this time gap. The TSX aborts the transaction when an application tries to access an unknown location or unknown target address. As these side-channel attacks use TSX, no software interrupts are triggered in the system and hence these kinds of attacks go undetected. The work in [131] demonstrates a technique by which the aforementioned attacks can be checked. Intel's TSX system has specific counters (such as `RTM_RETIRED.ABORTED`) to signal an alarm when transactions are aborted very frequently. By monitoring this specific counter the attacks can be mitigated.

C. Conclusion

The information captured from the available on-chip HPCs can be utilized for detecting malicious activities on the hardware ranging from application-based malware to side-channel attacks. Despite the proven benefits, the existing HPCs might not be efficient for achieving 'perfect security'. As such, there is an emerging need to perform a better analysis of the information captured in the HPCs to expose the characteristics of specially crafted attacks.

VII. Conclusion and Future Directions

As we continue to build larger and more complex systems hardware-assistant security will play an increasingly important role in the future. However, current solutions face a number of challenges that we identified in the paper. On one hand, solutions must be available for wide-spread use, unlike for instance ARM TrustZone. On the other hand, none of the existing solutions can provide perfect security, as demonstrated by the various attacks shown in the past.

Providing comprehensive security is costly and might only be needed for some rare highly sensitive use-cases. To account for the different requirements hardware must become more flexible, allowing more control from software. For example, by selectively disabling resource sharing high-security software can prevent side-channel leakage while other software can retain the performance benefits of utilizing shared resources. Going one step further, fine-grain control over the hardware would ultimately allow to re-configure or even patch hardware when a new vulnerability is detected.

In addition, efficient usage of on-chip performance counters is beneficial towards security without adding additional overheads. The hardware performance counters can be utilized for both malware and side-channel attack detection. However, adversarially crafted malware or sophisticated side-channel attacks, calls for devising better HPCs and/or advanced analysis techniques for achieving perfect security.

References

[1] M. Abadi *et al.*, "Control-flow integrity," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2005.

[2] L. Davi, P. Koeberl, and A.-R. Sadeghi, "Hardware-assisted fine-grained control-flow integrity: Towards efficient protection of embedded systems against software exploitation," in *Proceedings of the 51st Annual Design Automation Conference*, 2014.

[3] F. Brasser *et al.*, "Regulating ARM TrustZone Devices in Restricted Spaces," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16, 2016.

[4] ARM Limited, "ARM Security Technology – Building a Secure System using TrustZone Technology," http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009.

[5] J. M. McCune *et al.*, "Trustvisor: Efficient tcb reduction and attestation," in *Security and Privacy, IEEE Symposium on*, 2010.

[6] F. McKeen *et al.*, "Innovative Instructions and Software Model for Isolated Execution," in *Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[7] M. Hoekstra *et al.*, "Using Innovative Instructions to Create Trustworthy Software Solutions," in *Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[8] I. Anati *et al.*, "Innovative Technology for CPU Based Attestation and Sealing," in *Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[9] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *USENIX Symposium on Operating Systems Design and Implementation*, 2014.

[10] F. Schuster *et al.*, "VC3: Trustworthy data analytics in the cloud using SGX," in *IEEE Symposium on Security and Privacy*, 2015.

[11] S. Arnautov *et al.*, "SCONE: Secure linux containers with intel SGX," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016.

[12] S. Shinde *et al.*, "Panoply: Low-tcb linux applications with SGX enclaves," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2017.

[13] P. Koeberl *et al.*, "Time to Rethink: Trust Brokerage Using Trusted Execution Environments," in *Trust and Trustworthy Computing*, 2015.

[14] K. A. Küçük *et al.*, "Exploring the Use of Intel SGX for Secure Many-Party Applications," in *System Software for Trusted Execution (SysTEX)*, 2016.

[15] R. Bahmani *et al.*, "Secure Multiparty Computation from SGX," in *Financial Cryptography and Data Security (FC)*, 2017.

[16] O. Ohrimenko *et al.*, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security Symposium*, 2016.

[17] S. Chandra *et al.*, "Securing Data Analytics on SGX with Randomization," in *European Symp. on Research in Computer Security*, 2017.

[18] T. Hunt *et al.*, "Chiron: Privacy-preserving Machine Learning as a Service," *CoRR*, vol. abs/1803.05961, 2018.

[19] F. Brasser *et al.*, "Voiceguard: Secure and private speech processing," 2018.

[20] R. Maes, "Physically unclonable functions: Constructions, properties and applications," 2012.

[21] F. Armknecht *et al.*, "Towards a unified security model for physically unclonable functions," in *Cryptographers' Track at the RSA Conf.*, 2016.

[22] G. T. Becker, "The gap between promise and reality: On the insecurity of xor arbiter pufs," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2015.

[23] F. Ganji *et al.*, "Strong machine learning attack against pufs with no mathematical model," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[24] S. Wei *et al.*, "Reverse engineering and prevention techniques for physical unclonable functions using side channels," in *Design Automation Conference*, 2014.

[25] S. Zeitouni *et al.*, "Remanence decay side-channel: The puf case," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2016.

[26] S. Tajik *et al.*, "Physical characterization of arbiter pufs," in *Int. Conf. on Cryptographic Hardware and Embedded Systems*, 2014.

[27] U. Rührmair *et al.*, "Efficient power and timing side channels for physical unclonable functions," in *International Conference on Cryptographic Hardware and Embedded Systems*, 2014.

[28] J. Delvaux and I. Verbauwhede, "Fault injection modeling attacks on 65 nm arbiter and ro sum pufs via environmental changes," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2014.

[29] S. Tajik *et al.*, "Laser fault attack on physically unclonable functions," in *Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2015.

[30] O. Willers *et al.*, "Mems-based gyroscopes as physical unclonable functions," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[31] A. Vijayakumar and S. Kundu, "A novel modeling attack resistant puf design based on non-linear voltage transfer characteristics," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[32] R. Kumar and W. Burleson, "On design of a highly secure puf based on non-linear current mirrors," in *IEEE International Symposium on Hardware Oriented Security and Trust*, 2014.

[33] A. Vijayakumar *et al.*, "Machine learning resistant strong puf: Possible or a pipe dream?" in *Hardware Oriented Security and Trust (HOST)*, 2016.

[34] S. C. Konigsmark *et al.*, "Cnpuf: A carbon nanotube-based physically unclonable function for secure low-energy hardware design," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014.

[35] G. S. Rose and C. A. Meade, "Performance analysis of a memristive crossbar puf design," in *Design Automation Conference*, 2015.

[36] X. Zhang *et al.*, "A novel puf based on cell error rate distribution of stt-ram," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[37] Trusted Computing Group, "TPM 1.2 Protection Profile," https://www.trustedcomputinggroup.org/tpm-1-2-protection-profile/, 2016.

[38] *Intel Trusted Execution Technology MLE Developer's Guide*, Intel Corp., Dec. 2009.

[39] *AMD64 Virtualization Codenamed "Pacifica" Technology - Secure Virtual Machine Architecture Reference Manual*, AMD, 2005.

[40] J. M. McCune *et al.*, "Flicker: An Execution Infrastructure for Tcb Minimization," in *3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, ser. Eurosys, 2008.

[41] H. Shacham, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2007.

[42] S. Checkoway *et al.*, "Return-oriented programming without returns," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2010.

[43] K. Z. Snow *et al.*, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *34th IEEE Symposium on Security and Privacy*, ser. S&P, 2013.

[44] M. Conti *et al.*, "Losing control: On the effectiveness of control-flow integrity under stack attacks," in *ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS, 2015.

[45] F. Schuster *et al.*, "Counterfeit object-oriented programming: On the difficulty of preventing code reuse attacks in C++ applications," in *36th IEEE Symposium on Security and Privacy*, ser. S&P, 2015.

[46] R. Rudd *et al.*, "Address-oblivious code reuse: On the effectiveness of leakage-resilient diversity," in *Proceedings of the Network and Distributed System Security Symposium*, ser. NDSS, 2017.

[47] ARM Limited, "Security technology: building a secure system using TrustZone technology," http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2008.

[48] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The untapped potential of trusted execution environments on mobile devices," *IEEE Security & Privacy*, 2014.

[49] Dan Rosenberg, "Reflections on trusting trustzone," https://www.blackhat.com/docs/us-14/materials/us-14-Rosenberg-Reflections-on-Trusting-TrustZone.pdf, 2014.

[50] C. Josh Thomas, Nathan Keltner, "Reflections on Trusting TrustZone," https://pacsec.jp/psj14/PSJ2014_Josh_PacSec2014-v1.pdf, 2014.

[51] S. LLC, "Trustnone," http://theroot.ninja/disclosures/TRUSTNONE_1.0-11282015.pdf, 2015.

[52] Gal Beniamini, "QSEE Privilege Escalation Vulnerabilitiy," http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html, 2016.

[53] L. Aronsky, "KNOXout - bypassing samsung KNOX," 2016.

[54] N. Stephens, "Behind the PWN of a trustzone," https://www.slideshare.net/GeekPwnKeen/nick-stephenshow-does-someone-unlock-your-phone-with-nose, 2016.

[55] Tencent, "Defeating Samsung KNOX with Zero Privilege," https://www.blackhat.com/docs/us-17/thursday/us-17-Shen-Defeating-Samsung-KNOX-With-Zero-Privilege-wp.pdf, 2017.

[56] Project Zero, "Lifting the HyperVisor," https://googleprojectzero.blogspot.de/2017/02/lifting-hyper-visor-bypassing-samsungs.html, 2017.

[57] Project Zero, "Trust Issues: Exploiting TrustZone TEEs," https://googleprojectzero.blogspot.com/2017/07/trust-issues-exploiting-trustzone-tees.html, 2017.

[58] D. Kuvaiskii *et al.*, "SGXBOUNDS: Memory safety for shielded execution," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys, 2017.

[59] J. Lee *et al.*, "Hacking in darkness: Return-oriented programming against secure enclaves," in *Proceedings of 26th USENIX Security Symposium*, ser. USENIX Security, 2017.

[60] A. Bittau *et al.*, "Hacking blind," in *35th IEEE Symposium on Security and Privacy*, ser. S&P, 2014.

[61] J. Seo *et al.*, "SGX-Shield: Enabling address space layout randomization for SGX programs," in *Network and Distributed System Security Symposium*, 2017.

[62] A. Biondo *et al.*, "The guard's dilemma: Efficient code-reuse attacks against intel sgx," in *Proceedings of 27th USENIX Security Symposium*, ser. USENIX Security, 2018.

[63] M. Abadi *et al.*, "Control-flow integrity principles, implementations, and applications," *ACM Trans on Information System Security*, 2009.

[64] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *IEEE Symposium on Security and Privacy*, 2015.

[65] S. Lee *et al.*, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," in *USENIX Security Symposium*, 2017.

[66] M.-W. Shih *et al.*, "T-SGX: Eradicating controlled-channel attacks against enclave programs," in *Network and Distributed System Security Symposium*, 2017.

[67] S. Chen *et al.*, "Detecting privileged side-channel attacks in shielded execution with Déjá Vu," in *ACM Symposium on Information, Computer and Communications Security*, 2017.

[68] M. Schwarz *et al.*, "Malware Guard Extension: Using SGX to Conceal Cache Attacks," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2017.

[69] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," arXiv:1703.06986 [cs.CR], Tech. Rep., 2017, https://arxiv.org/abs/1703.06986.

[70] J. Götzfried *et al.*, "Cache Attacks on Intel SGX," in *European Workshop on Systems Security*, 2017.

[71] F. Brasser *et al.*, "Software grand exposure: SGX cache attacks are practical," in *USENIX Workshop on Offensive Technologies*, 2017.

[72] Y. Yarom, D. Genkin, and N. Heninger, "CacheBleed: A timing attack on OpenSSL constant time RSA," Cryptology ePrint Archive. Report 2016/224, Tech. Rep., 2016, https://eprint.iacr.org/2016/224.pdf.

[73] A. Moghimi, T. Eisenbarth, and B. Sunar, "Memjam: A false dependency attack against constant-time crypto implementations in sgx," in *Topics in Cryptology – CT-RSA 2018*, 2018.

[74] L. Domnitser *et al.*, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2012.

[75] D. Page, "Partitioned cache architecture as a side-channel defence mechanism," in *IACR Eprint archive*, 2005.

[76] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *Annual IEEE/ACM International Symposium on Microarchitecture*, 2008.

[77] L. Domnitser *et al.*, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization*, 2012.

[78] G. Keramidas *et al.*, "Non deterministic caches: A simple and effective defense against side channel attacks," *Design Automation for Embedded Systems*, 2008.

[79] F. Liu and R. B. Lee, "Random fill cache architecture," in *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[80] J. Kong *et al.*, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *IEEE International Symposium on High Performance Computer Architecture*, 2009.

[81] D. Gruss *et al.*, "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory," in *26th USENIX Security Symposium*, 2017.

[82] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Annual ACM Symposium on Theory of Computing*, 1987.

[83] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, 1996.

[84] E. Stefanov *et al.*, "Path ORAM: an extremely simple oblivious RAM protocol," in *ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[85] L. Ren *et al.*, "Constants count: Practical improvements to oblivious RAM," in *USENIX Security Symposium*, 2015.

[86] M. T. Goodrich *et al.*, "Privacy-preserving group data access via stateless oblivious RAM simulation," in *Annual ACM-SIAM symposium on Discrete Algorithms*, 2012.

[87] P. Williams and R. Sion, "Round-optimal access privacy on outsourced storage," 2012.

[88] M. Maas *et al.*, "Phantom: Practical oblivious computation in a secure processor," in *ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[89] C. Liu, M. Hicks, and E. Shi, "Memory trace oblivious program execution," in *IEEE Computer Security Foundations Symposium*, 2013.

[90] C. Liu *et al.*, "Ghostrider: A hardware-software system for memory trace oblivious computation," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, 2015.

[91] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *USENIX Security Symposium*, 2015.

[92] F. Brasser *et al.*, "DR.SGX: hardening SGX enclaves against cache attacks with data location randomization," *CoRR*, vol. abs/1709.09917, 2017. [Online]. Available: http://arxiv.org/abs/1709.09917

[93] S. Sasy, S. Gorbunov, and C. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," *IACR Cryptology ' Archive*, vol. Report 2017/549, 2017.

[94] B. Fuhry *et al.*, "HardIDX: Practical and Secure Index with SGX," in *Data and Applications Security and Privacy XXXI*, 2017.

[95] M. Lipp *et al.*, "Meltdown," *ArXiv e-prints*, Jan. 2018.

[96] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," *ArXiv e-prints*, Jan. 2018.

[97] M. Lanteigne, "How rowhammer could be used to exploit weaknesses in computer hardware," https://www.thirdio.com/rowhammer.pdf, 2016.

[98] Y. Kim *et al.*, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *41st Annual International Symposium on Computer Architecture*, 2014.

[99] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," https://googleprojectzero.blogspot.de/2015/03/exploiting-dram-rowhammer-bug-to-gain.html, 2016.

[100] G. Chen *et al.*, "SGXPECTRE Attacks: Leaking Enclave Secrets via Speculative Execution," *arXiv preprint arXiv:1802.09085*, 2018.

[101] P. D. S. Manoj *et al.*, "A Q-Learning based self-adaptive I/O communication for 2.5D integrated many-core microprocessor and memory," *IEEE Trans. on Computers*, vol. 65, no. 4, pp. 1185–1196, Apr. 2016.

[102] P. D. S. Manoj, H. Yu, and K. Wang, "3D many-core microprocessor power management by space-time multiplexing based demand-supply matching," *IEEE Trans. on Computers*, vol. 64, no. 11, pp. 3022–3036, Nov 2015.

[103] H. Sayadi *et al.*, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," in *IEEE International Conference on Computer Design (ICCD)*, 2017.

[104] H. Sayadi *et al.*, "Customized machine learning-based hardware-assisted malware detection in embedded devices," *IEEE International Conference On Trust, Security And Privacy In Computing And Communications (IEEE TrustCom-18)*, 2018.

[105] J. Stangl, T. Loruenser, and P. D. S. Manoj, "A fast and resource efficient FPGA implementation of secret sharing for storage applications," in *ACM/EDAA/IEEE DATE*, 2018.

[106] J. Demme *et al.*, "On the feasibility of online malware detection with performance counters," in *International Symposium on Computer Architecture*, 2013.

[107] M. B. Bahador, M. Abadi, and A. Tajoddin, "HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition," in *International Conference on Computer and Knowledge Engineering*, 2014.

[108] B. Singh *et al.*, "On the detection of kernel-level rootkits using hardware performance counters," in *Asia Conference on Computer and Communications Security*, 2017.

[109] X. Wang and R. Karri, "Reusing hardware performance counters to detect and identify kernel control-flow modifying rootkits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, Mar 2016.

[110] C. Malone, M. Zahran, and R. Karri, "Are hardware performance counters a cost effective way for integrity checking of programs," in *ACM Workshop on Scalable Trusted Computing*, 2011.

[111] X. Wang *et al.*, "Hardware performance counter-based malware identification and detection with adaptive compressive sensing," *ACM Trans. Archit. Code Optim.*, vol. 13, no. 1, Mar 2016.

[112] M. Ozsoy *et al.*, "Malware-aware processors: A framework for efficient online malware detection," in *IEEE International Symposium on High Performance Computer Architecture*, 2015.

[113] M. Ozsoy *et al.*, "Hardware-based malware detection using low-level architectural features," *IEEE Trans. Comput.*, vol. 65, no. 11, Nov 2016.

[114] H. Sayadi *et al.*, "Ensemble learning for effective run-time hardware-based malware detection: A comprehensive analysis and classification," in *Design Automation Conference*, 2018.

[115] H. Sayadi *et al.*, "Comprehensive assessment of run-time hardware-supported malware detection using general and ensemble learning," in *ACM Computing Frontiers*, 2018.

[116] K. N. Khasawneh *et al.*, "Ensemble learning for low-level hardware-supported malware detection," in *Research in Attacks, Intrusions, and Defenses*, 2015.

[117] K. N. Khasawneh *et al.*, "EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers," 2018.

[118] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," *CoRR*, vol. abs/1403.1631, 2014.

[119] B. Zhou *et al.*, "Hardware performance counters can detect malware: Myth or fact?" in *Asia Conference on Computer and Communications Security*, 2018.

[120] X. Wang *et al.*, "ConFirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *IEEE/ACM International Conference on Computer-Aided Design*, 2015.

[121] X. Wang *et al.*, "Malicious firmware detection with hardware performance counters," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, Sep 2016.

[122] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *Research in Attacks, Intrusion and Defense*, 2016.

[123] S. Briongos *et al.*, "Cacheshield: Protecting legacy processes against cache attacks," *CoRR*, vol. abs/1709.01795, 2017.

[124] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, 2016. [Online]. Available: https://doi.org/10.1016/j.asoc.2016.09.014

[125] M. Chiappetta, "Quickhpc," https://github.com/chpmrc/quickhpc.

[126] M.-M. Bazm *et al.*, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in *IEEE International Conference on Fog and Mobile Edge Computing*, 2018.

[127] J. Nomani and J. Szefer, "Predicting program phases and defending against side-channel attacks using hardware performance counters," in *Workshop on Hardware and Architectural Support for Security and Privacy*, 2015.

[128] TrendMicro, "Detecting attacks that exploit meltdown and spectre with performance counters," https://blog.trendmicro.com/trendlabs-security-intelligence/detecting-attacks-that-exploit-meltdown-and-spectre-with-performance-counters/.

[129] M. Hiramatsu, "Kprobe-based event tracing," https://www.kernel.org/doc/Documentation/trace/kprobetrace.txt.

[130] C. Disselkoen *et al.*, "Prime+abort: A timer-free high-precision L3 cache attack using intel TSX," in *USENIX Security Symposium*, 2017.

[131] Endgame, "Detecting spectre and meltdown using hardware performance counters," https://www.endgame.com/blog/technical-blog/detecting-spectre-and-meltdown-using-hardware-performance-counters.