A Survey of Published Attacks on Intel SGX

Alexander Nilsson*†, Pegah Nikbakht Bideh*, Joakim Brorsson*‡§
{alexander.nilsson,pegah.nikbakht_bideh,joakim.brorsson}@eit.lth.se
*Lund University, Department of Electrical and Information Technology, Sweden
†Advenica AB, Sweden
‡Combitech AB, Sweden
§Hyker Security AB, Sweden

***Abstract*—Intel Software Guard Extensions (SGX) provides a trusted execution environment (TEE) to run code and operate sensitive data. SGX provides runtime hardware protection where both code and data are protected even if other code components are malicious. However, recently many attacks targeting SGX have been identified and introduced that can thwart the hardware defence provided by SGX. In this paper we present a survey of all attacks specifically targeting Intel SGX that are known to the authors, to date. We categorized the attacks based on their implementation details into 7 different categories. We also look into the available defence mechanisms against identified attacks and categorize the available types of mitigations for each presented attack.**

I. Introduction

Trusted Execution Environments (TEEs) create isolated environments where sensitive code can run with higher security level than the operating system. Intel Software Guard Extensions (SGX) is an example of a TEE. SGX utilizes enclaves to isolate execution environment from other applications, the operating system's kernel and the hypervisor. SGX can run arbitrary code on general hardware and is suitable for cloud environments where it isolates the running code and data from the untrusted environment.

Without TEE solutions such as SGX, virtualization techniques are the primary defence that can be used by software to isolate code and data from other running software on a computer. Unfortunately virtualization techniques requires the application to trust the OS-kernel and hypervisor, and by extension the cloud provider in such a scenario.

SGX is only one of a few attempts at solving the issue of trusted computing in the cloud. Another solution is the Trusted Platform Module (TPM). The TPM however requires a larger chain of trust which is a drawback since it would require that the user roots its trust in the both the intentions of the implementers and in the absence of bugs in very large pieces of code (often including the BIOS', the OS kernel's and hypervisor's code-bases). Comparatively, SGX provides a great advantage, in that the root of trust is based only on the application code itself and the hardware implementation of the CPU.

Unfortunately, a relatively large number of flaws and attacks against SGX have been published by researchers over the last few years.

A. Contribution

In this paper, we present the first comprehensive review that includes all known attacks specific to SGX, including controlled channel attacks, cache-attacks, speculative execution attacks, branch prediction attacks, rogue data cache loads, microarchitectural data sampling and software-based fault injection attacks. For most of the presented attacks, there are countermeasures and mitigations that have been deployed as microcode patches by Intel or that can be employed by the application developer herself to make the attack more difficult (or impossible) to exploit. For all of the surveyed attacks in this paper, any known and relevant mitigation techniques are also presented.

B. Organization

In sec. II some background information on SGX is presented. The known attacks with their categorizations are given in sec. III. Then, the available mitigation techniques to categorized attacks are given in sec. IV. Finally, the current status of mitigation techniques and their applicability against specific attacks are discussed in sec. V and the paper is concluded in sec. VI.

II. Background on SGX

SGX is a set of extensions that aim to provide integrity and confidentiality for secure computations on computer systems where privileged software is potentially malicious.

SGX provides execution environments called enclaves to run code and operate sensitive data, where both code and data are protected from the outside software environment. This includes other applications running on the system and the operating system's kernel. Even the hypervisor, if it is running, is an actor from which SGX enclaves are protected. Notably, physical attacks are not considered in Intel's threat model, nor are so-called side-channel attacks.

For the rest of this section we refer to [1] without explicitly writing it out on each paragraph. We refer to it also for the interested reader who wishes a more detailed explanation on the internals of SGX.

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation

A. SGX Overview

The Intel x86 64-bit instruction set architecture (ISA), to which we limit the scope of this paper, defines several architectural privilege levels, each one strictly more capable than the one below it. The least privileged is ring 3 where all user-space applications run. This is the majority of the software on a running system[1]. Ignoring ring 2 and 1, which are not used by any major Operating System today, the next-most powerful privilege level is ring 0 in which the OS kernel is running. Software running in ring 0 is responsible for resource allocation, device management, context switching, page swapping and so forth.

Intel Virtual Machine Extensions (VMX) is a set of hardware virtualization instructions which introduces the additional privilege levels of VMX root and VMX non-root. Hypervisors usually run as VMX root in ring 0 and carry the ultimate responsibility of resource allocation.

This relates to Intel SGX in the following way: An SGX enclave always runs as ring 3 like any normal user-space application (either VMX root or non-root). Also like any normal user-space application it relies on the OS-kernel (ring 0) software for services such as scheduling, page swapping and hardware interrupt handling. This is despite the fact that none of the system software (ring 3 or 0) is trusted by the enclave threat model. This has been achieved by a rather complex series of hardware extensions as well as the exclusion of denial-of-service from the threat model. This is reasonable since protection against denial of service in an untrusted environment would be very hard to achieve, if not down-right impossible.

The code and data for all enclaves on a running system resides in the *Enclave Page Cache* (EPC) inside the *Processor Reserved Memory* (PRM) which is a reserved subset of the physical address space (DRAM). It is worth noting that this address range is protected by the CPU so that Direct Memory Access (DMA) is prohibited and that not even code running in the so-called Software Management Mode[2] can get access to its contents. In order to protect against snoops of external memory reads and writes the PRM is transparently encrypted and integrity protected before entering/exiting the memory bus. This means that the CPU package itself is the only place where enclave data can be read in its decrypted form.

Enclaves are designed to operate much like dynamic loadable modules[3] which are loaded directly into the virtual address space of user-space applications. This means that enclaves can be entered in much the same ways that API-calls are made into software libraries (although it is a more expensive operation). This makes it comparatively easy to modify existing programs.

Enclaves can only be entered at well defined entry points (much like a library API) as specified by the enclave author. This prevents memory mapping attacks and security check bypasses. While application software cannot access the memory space of the enclave the reverse is not true. The enclave have no restrictions in regards to the rest of the applications code and data, this facilitates easy and secure communication between the two modes.

The SGX design and implementation is fully backward compatible with other ISA extensions such as VMX which enables the use of this technology by cloud tenants where several virtual machines are co-hosted on the same hardware.

B. The SGX Lifetime

The enclave's lifetime is managed by the (untrusted) OS-kernel, this includes handling of page swapping, interrupts and CPU core scheduling. This is facilitated by several new instructions introduced by the SGX extensions. Some of the more important ones will be discussed in this section.

1) Creation: In order to create an enclave, ring 0 first issues the privileged ECREATE instruction. Enclave creation is intended as a service for applications, provided by the system software. The ECREATE instruction allocates a special page for the enclave called the SECS, like all other enclave pages it is located inside the protected PRM range. The SECS stores meta data for the enclave and it is critical for the enclave's security.

2) Loading: After creation the SECS is still marked as *uninitialized*. Only while the SECS is marked as such can EADD and EEXTEND instructions be issued for that enclave. These instructions are also privileged and can only be issued by ring 0. EADD is used to add pages into the protected virtual address space of the enclave.

EEXTEND is used for measuring data and code for software attestation. Attestation will be briefly discussed in sec. II-C.

3) Initialization: The OS-kernel in ring 0 must issue the EINIT instruction in order to initialize the enclave. However, before it can do that it must first obtain an EINIT Token Structure. The procedure for this is to utilize a special *Launch Enclave* (LE) which is signed by a special key whose corresponding public part is hardcoded into the SGX implementation by Intel.

4) Teardown: Ring 0 can issue the EREMOVE instruction to remove enclaves. This deallocates the specified page after it is made sure that no logical processor currently owns it. After the SECS page is deallocated the enclave is completely destroyed. EREMOVE refuses to deallocate the SECS before all other pages have been deallocated.

5) Synchronous Entry: Each logical processor executing the enclave code uses a Thread Control Structure (TCS) which controls the execution and makes sure that no two processors use the same TCS at the same time.

Disregarding the possibility of interrupts an enclave is executed as a controlled jump into the enclave's code by issuing the EENTER instruction. EENTER can only jump to predefined addresses which prevents a malicious host application from bypassing security checks that the enclave author might wish to perform. When entering enclave mode some

[1] This includes large parts of the OS as well, where the kernel is the obvious exception.

[2] An even higher privilege level than ring 0 and VMX-root, used solely by the motherboard firmware to manage, for example, the booting stage, fan control, power and sleep functions.

[3] Such as .dll files for PE and Windows based systems and .so files for ELF and Unix derived systems.

Não sei se isto realmente faz parte
do lifecycle, o artigo 1 não fala

registers are saved in order to be restored later when the enclave is done executing. While executing enclave code, the logical processor is said to be in *enclave mode*.

6) Synchronous Exit: One of two ways of exiting enclave mode is via the EEXIT instruction which additionally performs a restore of the registers saved by EENTER.

7) Asynchronous Exit: If a hardware exception occurs, such as an interrupt or fault while a logical processor is executing enclave code an AEX instruction is issued by the enclave before invoking the system software's default exception handler. This instruction saves the current execution context and restores the state saved by EENTER.

8) Resumption: Once the registered software handler for a hardware interrupt has finished, it jumps back to the asynchronous exit handler in the enclaves host's process. This handler is responsible for issuing the ERESUME instruction which puts the logical processor back into enclave mode which continues the execution which was interrupted.

9) Page Eviction and Reloading: The SGX eviction implementation relies on the privileged EWB instruction (restricted to ring 0) which encrypts and integrity protects the specified page with a symmetric key known only to the enclave. It also utilizes a mechanism for ensuring freshness, based on nonces. After the EWB instruction has evicted the PRM page, the system's default page-swapping mechanism can take over and flush the page to disk, if desired.

C. Software Attestation

The goal of software attestation is to verify that the software application running inside an enclave is trustworthy. This verification can be done by using a remote party. Attestation data within the software can be signed by requesting the SGX hardware implementation to generate an attestation signature. The signature can be used to uniquely identify the software and any optional data inside the enclave. The verifier can use the signature to make sure that the attestation data was generated by a specific software running on a genuine SGX implementation.

III. Attacks on SGX

We found a large group of attacks on SGX (most of them side-channel based) and using their implementation details, we divide them into 7 different categories. The categories and the attacks in each category are described below. The surveyed papers are mostly self-categorized, we have simply unified the most common terminology based on their technical details. A summary of the attacks presented in this section is provided in Table I.

Attacks that are out-of-scope: As mentioned, we consider denial-of-service attacks to be out-of-scope. Due to this we do not include for example attacks such as SGX-Bomb [2], where the CPU can be forced to shutdown due to a hardware bug called RowHammer [3], [4].

The security of SGX enclaves can be undermined if enclaves do not take care to strictly adhere to an agreed-upon secure interface between the trusted and untrusted code bases. To help facilitate this, a number of SDKs are available, to aid the construction of secure enclaves. In particular they help out with, for example, CPU status flag sanitation, correct stack pointer restoration, range checks of pointers and arrays and prevention of register leakage on exit. Of course, any vulnerability in the SDK itself will automatically impact all enclaves that make use of it. In "A Tale of Two Worlds" [5] Van Bulck et al. discovered multiple vulnerabilities in all open source SDKs for enclave development that they tested. This paper do not invalidate the security properties of SGX in and of itself, but it highlights the difficulty in writing secure software in general and enclaves in particular.

Another class of attacks are those that targets specific vulnerabilities of the enclave developer's own implementation. We mention a few such attacks here, for the readers convenience, but the list is by no means complete.

Checkoway and Shacham proposed the Iago attack [6]. The authors proposed to take advantage of the implicit trust applications place on the kernel, despite it being explicitly stated to exist outside the TCB. Conceptually the Iago attack uses a malicious OS-kernel to send false responses to system calls in order to fool the application under attack to perform operations against its own interests.

Lee et al. evaluated the old attack technique "Return-Oriented Programming" (ROP) in [7] and found that it is indeed possible to circumvent the hardware protections provided by the SGX design and achieve a total security break of the attacked enclave.

An attack was also introduced by Weichbrodt et al. [8] which exploits synchronization bugs in multi-threaded SGX enclaves. The authors built an attack tool called "AsyncShock" which simplifies the reliable exploitation of such bugs in enclave code.

A. Controlled Channel Attacks [9]–[14]

This section introduces the notion of *controlled channel attacks*, a term that Xu et al. [9] coined in 2015. It is a type of side-channel attack that make use of the near-total control the untrusted OS-kernel has over the platform. This control can be used to construct powerful side channels against the protected enclave who relies on the kernel's services. There have been several more attacks, based on the same principles.

The strategy used in [9], [10] was to monitor memory accesses with page-level granularity by monitoring or introducing page-faults. SGX-Step [11] instead configures APIC timers, issues interrupts and tracks page-table entries in such a way that it allows for single-stepping enclave code instructions.

SGX-Step is also used as a framework in many other attacks (see [5], [12], [13], [28], [34], [44]).

In [12] a timing-based side channel with instruction-level granularity is achieved by timing carefully synchronized interrupts while the enclave is running. The authors in [13] propose a side channel by exploiting the memory segmentation feature only available for code running in the 32-bit legacy mode.

Wang et al. proposed a new attack called *sneaky page monitoring* [14] which does not require any interrupts of the enclave by periodically accessing and resetting the *accessed* flag in the translation lookaside buffer, TLB.

B. Cache-attacks [20]–[25]

There have been many cache-based timing attacks against SGX enclaves published in the literature. Common to all of them are the exploitation of the cache-hierarchy system and the fact that the caching of memory loads from DRAM leaves effects in the system state which are measurable from outside the protected application. What these attacks show is that SGX enclaves are vulnerable to the same cache attacks against secret dependent information processing as any software application. In-fact they appear to be even more vulnerable due to the increased capability of the attackers in SGX's attack model.

There exist a number of different general (non SGX-specific) techniques for extracting information from side channels, we mention here *Flush+Reload* [15], *Prime+Probe* [16], *Evict+Time* [17], *Evict+Reload* [18] and *Flush+Flush* [19].

In the *sneaky page monitoring*[4] Wang et al. [14] explores several different ways of improving their attack. The authors particularly makes use of the Prime+Probe cache timing technique to increase the granularity of their attack.

The *CacheZoom* attack [20], introduced by Moghimi et al. also makes use of the Prime+Probe technique, as does Götzfried et al. [21] in their attack. Schwarz et al. [22] construct a malicious enclave from which they mount a Prime+Probe attack against other enclaves. Brasser et al. proposed a same-core attack (using HT) against the L1 cache [23], also using Prime+Probe. Prime+Probe is again used by Dall et al. [24] to attack Intel's *provisioning enclave* and thereby allows Intel themselves to break EPID's[5] unlinkability property.

The *MemJam* [25] attack by Moghimi et al. uses read-after-write false dependencies due to the 4K aliasing of the L1 cache. The methodology itself resembles that of Evict+Time.

C. Branch Prediction Attacks [26]–[28]

Lee et al. introduced the Branch Shadowing attack in [26] to reveal fine-grained control flow of a running SGX enclave, they showed that this could be used to break the security of several enclave-based constructs. In [27] Evtyushkin et al. proposed a similar attack, dubbed BranchScope, which uses the directional branch predictor instead of the BTB (branch target buffer) which is a companion component to the BTB. This shows that the branch predictor unit can be vulnerable even in the face of BTB protections. Bluethunder [28] is another branch predictor attack similar to BranchScope. The main difference of Bluethunder to BranchScope is that Bluethunder uses a 2-level directional predictor which is a completely different branch predictor unit. As a result Bluethunder is 52 times faster than BranchScope.

D. Speculative Execution Attacks [31]–[33]

Early 2018 the Spectre [29] and Meltdown [30] attacks made headlines outside the academic world. In this section, the Spectre attack in particular is of interest. This attack originally had 2 variants: bounds check bypass and branch target injection. The second variant targets the BTB, branch target buffer, in such a way that when the victim process executes an indirect branch instruction it mispredicts and speculatively executes code that never would have been executed. Of course, once the CPU-pipeline catches up and realizes that it was a misprediction it discards any results. Central to the Spectre attack however, is the hardware vulnerability that these speculatively executed instructions results in measurable changes to the CPU state, or in this case the CPU cache. In short, the Spectre attack allows an attacking process to infer some data values from vulnerable co-hosted processes.

In [31] Chen et al. answered the question of whether or not SGX is vulnerable to the Spectre or Spectre-like attacks in the affirmative. The authors presented the SgxPectre attack and used it to extract the secret seal keys and attestation keys from Intel signed quoting enclaves. In [32] Koruyeh et al. proposed the SpectreRSB attack which alternatively uses the return stack buffer which is a structure in modern CPUs used to speculatively predict the return address of execution frames (functions). SgxSpectre in [33] (not to be confused with SgxPectre [31]) also demonstrated a successful attack on SGX enclaves using a slight modification of the Spectre variant 1 attack.

E. Rogue Data Cache Loads [34]

Similarly to the Spectre-type of attacks Meltdown [30] exploits the out-of-order execution of modern CPUs. Unlike Spectre however, Meltdown does not explicitly make use of the speculative execution feature or the BTB, instead it relies on a race condition where the results of unauthorized memory accesses are transiently available for out-of-order executed instructions before the CPU issues a fault and rolls-back the results of these instructions. This implicitly affects the CPU cache and allows the memory access to be inferred.

SGX however, works slightly differently in that it does not issue any faults for accessing enclave memory, but instead uses *abort page semantics* [1] which allows the access with a dummy -1 result. The Foreshadow attack [34] works around this lack of race condition by instead relying on the fact that the abort page semantics applies only after normal page-table based permission checks succeeds without issuing a page-fault. Foreshadow therefore revokes all access to the enclave pages that it wishes to read using the mprotect system call, after this the principle of the Meltdown approach can again be used. Due to details of the SGX implementation this only works for memory which has been already cached in the first level cache (L1), Intel therefore categorized the Foreshadow vulnerability as a "L1 Terminal Fault" (L1TF) bug. [34]

In Foreshadow-NG [35] the authors generalize the L1TF attack into three versions: Foreshadow-SGX (original Foreshadow attack), Foreshadow-OS and Foreshadow-VMM. These attacks were also used in [36]. The latter 2 attacks were also discovered by Intel and are not applicable to SGX.

F. Microarchitectural Data Sampling (MDS) [38]–[41]

In late 2019 three papers of similar nature were published, namely *Fallout* [37], *RIDL* [38] and *ZombieLoad* [39]. Intel

[4] The basic attack is a controlled channel attack, see sec. III-A.

[5] EPID or Enhanced Privacy ID is Intel's recommended algorithm used for attestation while preserving privacy of the trusted system

has dubbed this new class of attacks *Microarchitectural Data Sampling* attacks, or MDS-attacks and they can be used to bypass most of the common security boundaries, such as: JavaScript sandboxes, processes, kernels, VMs and SGX enclaves. Although similar in nature to both Spectre and Meltdown, due to their use of out-of-order and speculative execution features they have another common theme; these works are based upon leakage of information from a number of implementation specific and undocumented intermediary buffers of the targeted micro architecture. Closely following these original papers *CacheOut* [40] *CrossTalk* [41] and *SX-Axe* [42] were published in the first half of 2020.

Fallout, for example, makes use of the *store buffer* to leak information of kernel writes to user space. Luckily, due to the flushing of the store buffer, SGX is safe from this particular attack. Unluckily, SGX still falls victim to the other attacks mentioned in the above paragraph.

RIDL, or *Rogue In-Flight Data Load*, and CacheOut both exploit the *Line Fill Buffers* (LFBs) to the effect of bypassing all — at their respective time of publication — deployed mitigations for Spectre, Meltdown and Foreshadow. The LFBs are used in the transfer paths between the L1 data caches (L1D) and the L2 caches. RIDL was the first work to critically analyse the behaviour of the LFBs from a MDS-attack perspective.

ZombieLoad also targets the LFBs[6], as the name suggests however, it leaks whatever *stale* data currently resides in the buffers. ZombieLoad is an improvement in that while RIDL only leaks data from loads *not currently* residing in the L1D cache, ZombieLoad leaks the results of memory loads, regardless of the requested data's presence in L1D or not.

Both RIDL and ZombieLoad suffer from a "drinking from a firehose" [40] problem, in that they are unable to control which data is loaded into the LFBs and subsequently leaked. CacheOut later solved this problem by forcing contention on the L1D data it wishes to target, and thereby evicting it (through the LFBs) from the cache.

SGAxe [42] is not a new attack, per se, but it utilized the CacheOut attack to extract the sealing key and in turn the machine's attestation key from the Intel provided Quoting Enclave. This key could be used to forge attestation quotes.

CrossTalk [41] extends the MDS techniques to show that enclaves can be attacked even across different execution units (i.e. cross core). An otherwise good mitigation strategy is to isolate execution of sensitive operation to CPU cores not shared with untrusted threads. CrossTalk achieves this by managing to sample the so called staging buffer which is an undocumented component on modern Intel CPUs, shared between all cores. The paper demonstrates the attack by showing how snooping on the output from the rdrand instructions can be used to extract a ECDSA private key from a SGX enclave running on an separate core.

G. *Software-based Fault Injection Attacks [44]*

In 2017 the CLKscrew [43] by Tang et al. attacked ARM TrustZone by adjusting the dynamic frequency scaling, through a privileged and model-specific interface (used by system software for dynamic overclocking). This attack does not affect Intel SGX since it is specific to ARM based systems, but recently Murdock et al. published the Plundervolt attack [44] which does affect SGX.

The Plundervolt attack abuses privileged interfaces for dynamic voltage scaling on the x86 CPU in order to reliably corrupt enclave computations. The authors write "Using this interface to very briefly decrease the CPU voltage during a computation in a victim SGX enclave, we show that a privileged adversary is able to inject faults into protected enclave computations". The authors then proceeds to demonstrate how this attack can be used to "reconstruct full cryptographic keys with negligible computational effort". [44]

The SGX-Bomb [2] and RowHammer [3], [4] hardware bugs mentioned earlier would also fit in this category, if denial of service attacks had not been placed out of scope for this article.

IV. Defensive Strategies and Mitigations

In this section, we present the most relevant published mitigation techniques for the presented attacks and place each into the categories: Microcode patch, System design, Compiler/SDK and Application design.

Defences that are out of scope: This survey does not account for purely theoretical defenses and mitigations which relies on changes to hardware and ISA. Some examples are Sanctum [45] and Autarky [46] which both proposes changes to the hardware and ISA.

A. *Microcode patch*

A CPU is not fully realized in hardware but rather most of the more complex instructions are implemented by a form of low-level software called microcode. The microcode can only be changed by the manufacturer of the CPU, in this case Intel. A microcode patch can thus make direct changes in how the CPU performs its duties, and these patches are usually the most effective way to mitigate any vulnerability.

B. *System design*

Some attacks cannot easily be thwarted by microcode patches but may instead be fixed by redesigning or removing implementation issues of the supporting systems. For example, Intel provides a number of special enclaves to support higher level services such as the *Launcher Enclave* (LE), *Provisioning Enclave* (PE) and *Quoting Enclave* (QE).

Some microcode patches include new information of the running system, which the attestation services and supporting enclaves must act upon, or otherwise include in the attestation reports [47].

C. *Compiler/SDK*

In some cases Intel does not appear to provide any solutions or mitigations for attacks, but leaves the responsibility up to the enclave authors themselves. In this case the best one can hope for is for some kind of general approach implemented

[6] at least in part, although the authors speculate on other sources of the leakage as well

TABLE I

SUMMARY OF ALL CITED ATTACKS LISTED HERE WITH A NUMBER OF PROPERTIES DISPLAYED IN A TABLE-FORMAT. HERE WE USE ●, ◐ AND ○ TO MEAN YES, PARTLY AND NO, RESPECTIVELY.

				Impact						Mitigations			
Attacks (abbreviated titles)	Type	SGX Specific	Targeted attack	Page access pattern	Instruction trace	Instruction latency	Memory access pattern	Memory Contents	Fault Injection	Microcode patch	System design	Compiler/SDK	Application design
Controlled-Channel [9]	sec. III-A	●	●	●	○	○	○	○	○	○	○	●[52]	●[55]
Stealthy Page Table [10]	sec. III-A	●	●	●	○	○	○	○	○	○	○	●[52]	○
SGX-Step [11]	sec. III-A	●	○	●	●	●	○	○	○	○	○	●[11] [48]	○
Nemesis [12]	sec. III-A	●	●	○	○	●	○	○	○	○	○	●[50]	○
Off Limits [13]	sec. III-A	●	●	●	◐	○	◐	○	○	●[13]	○	○	●[13]
Leaky Cauldron [14]	sec. III-B	●	●	●	○	○	●	○	○	○	○	●[50] [51]	○
CacheZoom [20]	sec. III-B	●	●	○	○	○	●	○	○	○	○	●[50] [51]	●[56]
Cache Attacks on SGX [21]	sec. III-B	●	●	○	○	○	●	○	○	○	○	○	●[56]
Malware Guard Extensions [22]	sec. III-B	●	●	○	○	○	●	○	○	○	●[22]	○	○
Software Grand Exposure [23]	sec. III-B	●	●	○	○	○	●	○	○	○	○	○	●[23]
CacheQuote [24]	sec. III-B	●	●	○	○	○	●	○	○	○	●[24]	○	○
MemJam [25]	sec. III-B	○	●	○	○	○	●	○	○	○	○	●[49]	○
Branch Shadowing [26]	sec. III-C	●	●	○	●	○	○	○	○	○	○	●[26] [48]	○
BranchScope [27]	sec. III-C	○	●	○	●	○	○	○	○	○	○	●[27] [48]	○
Bluethunder [28]	sec. III-C	●	●	○	●	○	○	○	○	○	○	●[28]	○
SgxPectre [31]	sec. III-D	●	●	○	○	○	○	●	○	●[47]	○	○	○
SpectreRSB [32]	sec. III-D	○	●	○	○	○	○	●	○	○	○	○	●[32]
Spectre v1 [33]	sec. III-D	○	●	○	○	○	○	●	○	●[47]	○	○	○
Foreshadow-SGX [34]	sec. III-E	●	○	○	○	○	○	●	○	●[35]	○	○	○
RIDL [38]	sec. III-F	○	●	○	○	○	○	●	○	●[38]	○	○	○
ZombieLoad [39]	sec. III-F	○	●	○	○	○	○	●	○	●[39]	○	○	○
CacheOut [40]	sec. III-F	●	●	○	○	○	○	●	○	◐[40]	○	○	○
CrossTalk [41]	sec. III-F	●	○	○	○	○	○	●	○	●[41]	○	○	○
Plundervolt [44]	sec. III-G	●	◐	○	○	○	○	○	●	●[44]	○	○	○

in either the compiler or in the enclave SDK. By going in this direction its use is, in theory, transparent to the enclave developer. That-is, if they elect to opt-in to the techniques and their various performance impacts and drawbacks. In this section, we will explore some known solutions that strive to fit in this category.

Lee et al. [26] proposed *ZigZagger* as a defence against their own branch shadowing attack. It works by transforming conditional branches into unconditional jumps to intermediate code sections (called trampolines) that in-turn bounces to the target code. Hosseinzadeh et al. [48] later improves upon this idea by doing control flow randomization at run-time. Both of these schemes were implemented as compiler extensions on top of LLVM. Meanwhile Chen et al. presents a solution also implemented in LLVM that closes HT (Hyper-Threading) based side-channels [49] by blocking access to sibling cores via the creation of a shadow thread[7].

Shih et al. presented a modified LLVM compiler dubbed T-SGX [50] which terminates the execution of sensitive operations using Intel's Transactional Synchronization Extensions, TSX, if a certain number of interrupts occurs. T-SGX is claimed to be effective against all known controlled channel

[7] Previously it would not help to disable Hyper-Threading since its status was not included in the attestation reports. But with the introduction of the Foreshadow mitigations, disabling HT in BIOS would now appear to be a more secure alternative.

attacks. “Déjà Vu” is an alternative solution proposed by Chen et al. in [51] which implements a clock-thread protected with TSX by which it can be detected if the run-time of the program differs significantly from what is expected. If it does, it is assumed that the protected code has been interrupted and Déjà Vu will therefore abort the execution.

Strackx et al. presented the “Heisenberg Defence” [52] as an alternative. It also utilizes TSX transactions, but by adding preloading and verifications it can proactively protect sensitive code against page-table based controlled channel attacks. For systems without TSX support, SGX-LAPD was proposed by Fu et al. in [53] using a detection based approached similar to T-SGX but instead implemented using large pages.

To defeat enclave specific attacks such as, for example, ROP attacks (which remain out of scope for this paper) Seo et al. [54] found a way to activate ASLR inside SGX enclaves, to make exploitation more difficult. It is implemented on top of the LLVM compiler.

D. *Application design*

If all else fails, the enclave’s author must take care to design their enclaves in a secure and side-channel protected manner. In this section we discuss some of the published tools and techniques that have been proposed to help with this.

Shinde et al. suggest a compiler *assisted* solution to remove data dependent memory accesses [55] in order to mitigate page-fault based side channels by aiming to keep secret data and code within the same page. Cloak is a software library developed by Gruss et al. in [56] that allows secret data and secret handling code to be wrapped in a TSX transaction. It seems to be quite effective at preventing cache-based side-channels.

An alternative solution is to introduce random noise in the applications algorithms by adding accesses to dummy data. Chandra et al. explores this possibility in [57]. The same effect can be more thoroughly obtained by the use of *Oblivious RAM* (ORAM) constructions with *data oblivious execution* to hide memory accesses in such a way that none-of the above mentioned side-channel attacks would be effective. ZeroTrace [58] introduced by Sasy et al. is one such scheme for use in SGX enclaves.

V. Discussion on the current status of mitigations

In table I, we have compiled a matrix that summarizes a number of properties for each attack, including the current state of mitigations, to the best of our knowledge.

As discussed earlier, the mitigation techniques are divided into four categories: Microcode patch, System design, Compiler/SDK and Application design. As it can be seen in Table I, almost all categorized attacks are mitigated or partially mitigated by these techniques.

Usually, the mitigation techniques for attacks in the same category are similar to each other. For instance, Controlled Channel Attacks [9]–[14] and Branch Prediction Attacks [26]–[28] can be mitigated using Compiler/SDK techniques or modifying the Application design.

The Cache-attacks [20]–[25] are explicitly outside the scope of the Intel threat-model. This, in combination with the fact that these attack are very specific for each targeted enclave implementation is the reason why one would be forced to make enclave specific mitigations (i.e. modifying the application design). In the case of the cache attack in [24], since it directly attacks part of the attestation service, the only viable option is for Intel to update the affected enclaves. At this time we have found no indication whether or not this has been done.

Most of the Speculative Execution Attacks, Rogue Data Cache Loads, MDS Attacks, and Software-based Fault Injection Attacks are mitigated using Microcode patch. This is the case for the 32-bit memory segmentation attack “Off-Limits” in [13], SgxPectre [31], Foreshadow [34], Foreshadow-NG [35] and PlunderVolt [44]. In a few instances the mitigation is in combination with updates to Intel provided attestation services [47].

While many of these mitigations appear to be quite effective, in practice some of them impose additional requirements on the system. For example, part of the mitigation strategy against Foreshadow would be to disable Hyper-Threading (HT) (logical cores share the same L1 cache) and for this reason the status of HT is included during attestation and sealing operations. These mitigations have been supplied through microcode-updates.

Koruyeh et al. mentions in SpectreRSB paper [32] that the microcode patch “RSB-refilling” is not specifically applied for SGX enclaves. Based on this we draw the conclusion that RSB-refilling ought to be implemented either the SDKs or compilers, if they wish to ensure protection against the SpectreRSB. We have been unable find any information on whether or not this or some other equivalent mitigation is implemented in the available SDKs. Alternatively, the enclave authors might wish to add this protection manually (this is shown in the table under the “Application Design” mitigation strategy).

For most MDS type of attacks, [38], [39] Intel microcode-patches had been already released, but for a more recent attack of this type called CacheOut [40] the update patch had not yet been released, at the time of this writing. Intel has promised [40] that the patch will be released in the near future. Until then, we mark this attack as partially mitigated in Table I.

The BIOS patch and microcode updates which mitigates PlunderVolt works by disabling much of the dynamic voltage scaling interface as well as recording and verifying the status of these interfaces into its sealing and attestation operations.

VI. Conclusions

We found more than 20 attacks in the literature using side channels and active attackers, see Table I. For all of the attacks there are mitigation strategies, however some are more feasible in practice than others. Especially those mitigations which are based on a correct application design might, in practice, be difficult to implement completely.

The number of side channel attacks targeting SGX found recently hint that there might be more attacks not yet discovered. This should also be considered when evaluating if and how to employ Intel SGX as a protection mechanism.

References

[1] V. Costan and S. Devadas, "Intel sgx explained.," *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[2] Y. Jang, J. Lee, S. Lee, and T. Kim, "SGX-bomb: Locking down the processor via rowhammer attack," in *SysTEX 2017 - 2nd Work. Syst. Softw. Trust. Exec. Coloca. with ACM SOSP 2017*, New York, New York, USA: ACM Press, 2017, pp. 1–6, ISBN: 9781450350976. DOI: 10.1145/3152701.3152709. [Online]. Available: https://dl.acm.org/citation.cfm?id=3152709http://dl.acm.org/citation.cfm?doid=3152701.3152709.

[3] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 361–372, 2014, ISSN: 01635964. DOI: 10.1145/2678373.2665726. [Online]. Available: https://dl.acm.org/citation.cfm?id=2665726http://dl.acm.org/citation.cfm?doid=2678373.2665726.

[4] M. Seaborn and T. Dullien, "Exploiting the DRAM rowhammer bug to gain kernel privileges," *BlackHat*, pp. 1–71, 2015. [Online]. Available: https://docs.huihoo.com/blackhat/usa-2015/us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdfhttp://secwiki.neu.edu.cn/wiki/images/a/a6/Us-15-Seaborn-Exploiting-The-DRAM-Rowhammer-Bug-To-Gain-Kernel-Privileges.pdfhttp://ieeexplore.ieee.org/document/6853210/.

[5] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia, and F. Piessens, "A Tale of Two Worlds," in *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS '19*, New York, New York, USA: ACM Press, 2019, pp. 1741–1758, ISBN: 9781450367479. DOI: 10.1145/3319535.3363206. [Online]. Available: https://people.cs.kuleuven.be/{~}jo.vanbulck/ccs19-tale.pdfhttp://dl.acm.org/citation.cfm?doid=3319535.3363206.

[6] S. Checkoway and H. Shacham, "Iago attacks: Why the system call API is a bad untrusted rpc interface," in *ACM SIGPLAN Not.*, vol. 48, 2013, pp. 253–263. DOI: 10.1145/2499368.2451145. [Online]. Available: https://hovav.net/ucsd/dist/iago.pdf.

[7] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, and B. B. Kang, "Hacking in Darkness: Return-oriented Programming against Secure Enclaves," *Proc. USENIX Secur. Symp. (USENIX Secur.*, pp. 523–539, 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk.

[8] N. Weichbrodt, A. Kurmus, P. Pietzuch, and R. Kapitza, "AsyncShock: Exploiting Synchronisation Bugs in Intel SGX Enclaves," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9878 LNCS, 2016, pp. 440–457, ISBN: 9783319457437. DOI: 10.1007/978-3-319-45744-4_22. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-45744-4{_}22http://link.springer.com/10.1007/978-3-319-45744-4{_}22.

[9] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proc. - IEEE Symp. Secur. Priv.*, vol. 2015-July, 2015, pp. 640–656, ISBN: 9781467369497. DOI: 10.1109/SP.2015.45. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7163052/.

[10] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling Your Secrets Without Page Faults: Stealthy Page Table-based Attacks on Enclaved Execution," *Proc. 26th USENIX Conf. Secur. Symp.*, pp. 1041–1056, 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulckhttp://dl.acm.org/citation.cfm?id=3241189.3241271.

[11] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-Step," in *Proc. 2nd Work. Syst. Softw. Trust. Exec. - SysTEX'17*, New York, New York, USA: ACM Press, 2017, pp. 1–6, ISBN: 9781450350976. DOI: 10.1145/3152701.3152706. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3152701.3152706.

[12] ——, "Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic," in *Proc. 2018 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS '18*, New York, New York, USA: ACM Press, 2018, pp. 178–195, ISBN: 9781450356930. DOI: 10.1145/3243734.3243822. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3243734.3243822.

[13] J. Gyselinck, J. Van Bulck, F. Piessens, and R. Strackx, "Off-Limits: Abusing Legacy x86 Memory Segmentation to Spy on Enclaved Execution," in *Springer*, 2018, pp. 44–60. DOI: 10.1007/978-3-319-94496-8_4. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-94496-8{_}4http://link.springer.com/10.1007/978-3-319-94496-8{_}4.

[14] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, "Leaky Cauldron on the Dark Land," in *Proc. 2017 ACM SIGSAC Conf. Comput. Commun. Secur. - CCS '17*, New York, New York, USA: ACM Press, 2017, pp. 2421–2434, ISBN: 9781450349468. DOI: 10.1145/3133956.3134038. arXiv: 1705.07289. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3133956.3134038http://arxiv.org/abs/1705.07289http://dx.doi.org/10.1145/3133956.3134038.

[15] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 719–732, ISBN: 9781931971157. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

[16] D. A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell.*

Lect. Notes Bioinformatics), vol. 3960 LNCS, 2006, pp. 1–20. DOI: 10.1007/11605805_1. [Online]. Available: http://link.springer.com/10.1007/11605805{_}1.

[17] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient Cache Attacks on AES, and Countermeasures," *J. Cryptol.*, vol. 23, no. 1, pp. 37–71, 2010, ISSN: 0933-2790. DOI: 10.1007/s00145-009-9049-y. [Online]. Available: https://link.springer.com/article/10.1007/s00145-009-9049-yhttp://link.springer.com/10.1007/s00145-009-9049-y.

[18] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *Proc. 24th USENIX Secur. Symp.*, vol. 897, 2015, pp. 897–912, ISBN: 9781931971232. [Online]. Available: www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/gruss.

[19] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+Flush: A Fast and Stealthy Cache Attack," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9721, Springer Verlag, 2016, pp. 279–299, ISBN: 9783319406664. DOI: 10.1007/978-3-319-40667-1_14. arXiv: 1511.04594. [Online]. Available: http://arxiv.org/abs/1511.04594http://link.springer.com/10.1007/978-3-319-40667-1{_}14.

[20] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10529 LNCS, pp. 69–90, 2017, ISSN: 16113349. DOI: 10.1007/978-3-319-66787-4_4.

[21] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on intel SGX," *Proc. Proc. 10th Eur. Work. Syst. Secur. EuroSec 2017, co-located with Eur. Conf. Comput. Syst. EuroSys 2017*, 2017. DOI: 10.1145/3065913.3065915.

[22] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using SGX to conceal cache attacks," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10327 LNCS, 2017, pp. 3–24, ISBN: 9783319608754. DOI: 10.1007/978-3-319-60876-1_1.

[23] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi, "Software Grand Exposure: SGX Cache Attacks Are Practical," *Usenix.org*, 2017. arXiv: 1702.07521. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasserhttp://arxiv.org/abs/1702.07521.

[24] F. Dall, G. D. Micheli, T. Eisenbarth, D. Genkin, N. Heninger, A. Moghimi, and Y. Yarom, "CacheQuote: Efficiently Recovering Long-term Secrets of SGX EPID via Cache Attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2018, no. 2, pp. 171–191, 2018, ISSN: 2569-2925. DOI: https://doi.org/10.13154/tches.v2018.i2.171-191. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/879.

[25] A. Moghimi, T. Eisenbarth, and B. Sunar, "MemJam: A False Dependency Attack against Constant-Time Crypto Implementations," *Int. J. Parallel Program.*, vol. 47, no. 4, pp. 538–570, 2017, ISSN: 0885-7458. DOI: 10.1007/s10766-018-0611-9. arXiv: 1711.08002. [Online]. Available: http://arxiv.org/abs/1711.08002http://dx.doi.org/10.1007/s10766-018-0611-9http://link.springer.com/10.1007/s10766-018-0611-9.

[26] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing," 2017. arXiv: 1611.06952. [Online]. Available: http://arxiv.org/abs/1611.06952.

[27] D. Evtyushkin, R. Riley, N. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," in *ACM SIGPLAN Not.*, vol. 53, New York, New York, USA: ACM Press, 2018, pp. 693–707, ISBN: 9781450349116. DOI: 10.1145/3173162.3173204. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3173162.3173204.

[28] T. Huo, X. Meng, W. Wang, C. Hao, P. Zhao, J. Zhai, and M. Li, "Bluethunder: A 2-level Directional Predictor Based Side-Channel Attack against SGX," *IACR Trans. Cryptogr. Hardw. Embed. Syst. 2020(1)*, DOI: https://doi.org/10.13154/tches.v2020.i1.321-347. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8401.

[29] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," in *2019 IEEE Symp. Secur. Priv.*, IEEE, 2019, pp. 1–19, ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00002. arXiv: 1801.01203. [Online]. Available: https://arxiv.org/abs/1801.01203https://ieeexplore.ieee.org/document/8835233/http://arxiv.org/abs/1801.01203.

[30] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *CFA Inst. Mag.*, vol. 18, no. 4, pp. 39–43, 2018, ISSN: 1543-1398. DOI: 10.2469/cfm.v18.n4.4762. arXiv: 1801.01207. [Online]. Available: https://arxiv.org/abs/1801.01207.

[31] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution," in *2019 IEEE Eur. Symp. Secur. Priv.*, IEEE, 2019, pp. 142–157, ISBN: 978-1-7281-1148-3. DOI: 10.1109/eurosp.2019.00020. arXiv: 1802.09085. [Online]. Available: http://arxiv.org/abs/1802.09085https://ieeexplore.ieee.org/document/8806740/http://dx.doi.org/10.1109/EuroSP.2019.00020.

[32] E. M. Koruyeh, K. Khasawneh, C. Song, and N. Abu-Ghazaleh, "Spectre Returns! Speculation Attacks using the Return Stack Buffer," *Usenix.org*, 2018. arXiv: 1807.07940. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/koruyehhttp://arxiv.org/abs/1807.07940.

[33] D. O'Keeffe, D. Muthukumaran, P.-L. Aublin, F. Kelbert, C. Priebe, J. Lind, H. Zhu, and P. Pietzuch, *Spectre attack against SGX enclave*, 2018.

[34] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution," *Proc. 27th {USENIX} Secur. Symp.*, 2018. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-van{_}bulck.pdf.

[35] O. Weisse, J. V. Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom, "Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution," *Proc. 27th USENIX Secur. Symp.*, vol. 0, 2018.

[36] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Breaking Virtual Memory Protection and the SGX Ecosystem with Foreshadow," *IEEE Micro*, vol. 39, no. 3, pp. 66–74, 2019, ISSN: 0272-1732. DOI: 10.1109/MM.2019.2910104. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8691527/https://ieeexplore.ieee.org/document/8691527/.

[37] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck, and Y. Yarom, "Fallout," in *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA: ACM, 2019, pp. 769–784, ISBN: 9781450367479. DOI: 10.1145/3319535.3363219. [Online]. Available: http://dl.acm.org/doi/10.1145/3319535.3363219.

[38] S. van Schaik, A. Milburn, S. Osterlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "RIDL: Rogue In-Flight Data Load," in *2019 IEEE Symp. Secur. Priv.*, IEEE, 2019, pp. 88–105, ISBN: 978-1-5386-6660-9. DOI: 10.1109/SP.2019.00087. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8835281/https://ieeexplore.ieee.org/document/8835281/.

[39] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "ZombieLoad," in *Proc. 2019 ACM SIGSAC Conf. Comput. Commun. Secur.*, New York, NY, USA: ACM, 2019, pp. 753–768, ISBN: 9781450367479. DOI: 10.1145/3319535.3354252. arXiv: 1905.05726. [Online]. Available: http://dl.acm.org/doi/10.1145/3319535.3354252.

[40] S. Van Schaik, M. Minkin, A. Kwong, D. Genkin, and Y. Yarom, "CacheOut: Leaking Data on Intel CPUs via Cache Evictions," *Cacheoutattack.com*, p. 16, 2020. [Online]. Available: http://cacheoutattack.com/CacheOut.pdf.

[41] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "CROSSTALK: Speculative Data Leaks Across Cores Are Real," in *Security and Privacy*, 2021.

[42] S. van Schaik, A. Kwong, D. Genkin, and Y. Yarom, *SGAxe : How SGX Fails in Practice*, 2020. [Online]. Available: https://sgaxe.com/.

[43] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," *26th USENIX Secur. Symp. (USENIX Secur. 17)*, pp. 1057–1074, 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.

[44] K. Murdock, D. Oswald, F. D. Garcia, J. V. Bulck, D. Gruss, and F. Piessens, "Plundervolt : Software-based Fault Injection Attacks against Intel SGX," *Proc. 41st IEEE Symp. Secur. Priv.*, 2020. [Online]. Available: https://doi.ieeecomputersociety.org/.

[45] V. Costan, I. Lebedev, and S. Devadas, "Sanctum : Minimal Hardware Extensions for Strong Software Isolation This paper is included in the Proceedings of the Sanctum : Minimal Hardware Extensions for Strong Software Isolation," *Usenix.org*, 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan.

[46] M. Orenbach, A. Baumann, and M. Silberstein, "Autarky: Closing controlled channels with self-paging enclaves," *Marksilberstein.com*, p. 16, 2020. DOI: 10.1145/3342195.3387541. [Online]. Available: https://doi.org/10.1145/3342195.3387541.

[47] C. Canella, J. Van Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin, and D. Gruss, "A Systematic Evaluation of Transient Execution Attacks and Defenses," *Usenix.org*, 2018. arXiv: 1811.05441. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/canellahttp://arxiv.org/abs/1811.05441.

[48] S. Hosseinzadeh, H. Liljestrand, V. Leppänen, and A. Paverd, *Mitigating Branch-Shadowing Attacks on Intel SGX using Control Flow Randomization*, New York, New York, USA, 2018. DOI: 10.1145/3268935.3268940. arXiv: 1808.06478. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3268935.3268940http://arxiv.org/abs/1808.06478http://dx.doi.org/10.1145/3268935.3268940.

[49] G. Chen, W. Wang, T. Chen, S. Chen, Y. Zhang, X. Wang, T.-H. Lai, and D. Lin, "Racing in Hyperspace: Closing Hyper-Threading Side Channels on SGX with Contrived Data Races," in *2018 IEEE Symp. Secur. Priv.*, vol. 2018-May, IEEE, 2018, pp. 178–194, ISBN: 978-1-5386-4353-2. DOI: 10.1109/SP.2018.00024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8418603/https://ieeexplore.ieee.org/document/8418603/.

[50] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs," March, 2017. DOI: 10.14722/ndss.2017.23193.

[51] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu," in *Proc. 2017 ACM Asia Conf. Comput. Commun. Secur. - ASIA CCS '17*, New York,

New York, USA: ACM Press, 2017, pp. 7–18, ISBN: 9781450349444. DOI: 10.1145/3052973.3053007. [Online]. Available: https://dl.acm.org/citation.cfm?id=3053007http://dl.acm.org/citation.cfm?doid=3052973.3053007.

[52] R. Strackx and F. Piessens, "The Heisenberg Defense: Proactively Defending SGX Enclaves against Page-Table-Based Side-Channel Attacks," *Arxiv.org*, 2017. arXiv: 1712.08519. [Online]. Available: http://arxiv.org/abs/1712.08519.

[53] Y. Fu, E. Bauman, R. Quinonez, and Z. Lin, "Sgx-Lapd: Thwarting Controlled Side Channel Attacks via Enclave Verifiable Page Faults," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10453 LNCS, Springer, Cham, 2017, pp. 357–380, ISBN: 9783319663319. DOI: 10.1007/978-3-319-66332-6_16. [Online]. Available: http://link.springer.com/10.1007/978-3-319-66332-6{_}16.

[54] J. Seo, B. Lee, S. Kim, M.-W. Shih, I. Shin, D. Han, and T. Kim, "SGX-Shield: Enabling Address Space Layout Randomization for SGX Programs," in *Proc. 2017 Netw. Distrib. Syst. Secur. Symp.*, Reston, VA: Internet Society, 2017, ISBN: 1-891562-46-0. DOI: 10.14722/ndss.2017.23037. [Online]. Available: https://gts3.org/assets/papers/2017/seo:sgx-shield.pdfhttps://www.ndss-symposium.org/ndss2017/ndss-2017-programme/sgx-shield-enabling-address-space-layout-randomization-sgx-programs/.

[55] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing Your Faults From Telling Your Secrets: Defenses Against Pigeonhole Attacks," *Arxiv.org*, 2015. arXiv: 1506.04832. [Online]. Available: https://arxiv.org/abs/1506.04832http://arxiv.org/abs/1506.04832.

[56] D. Gruss, J. Lettner, F. Schuster, O. O. Ohrimenko, I. Haller, M. Costa, and O. O. Ohrimenko, "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory," *Usenix Secur. 2017*, pp. 217–234, 2017. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/gruss.

[57] S. Chandra, V. Karande, Z. Lin, L. Khan, M. Kantarcioglu, and B. Thuraisingham, "Securing Data Analytics on SGX with Randomization," in *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10492 LNCS, Springer Verlag, 2017, pp. 352–369, ISBN: 9783319664019. DOI: 10.1007/978-3-319-66402-6_21. [Online]. Available: http://link.springer.com/10.1007/978-3-319-66402-6{_}21.

[58] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace : Oblivious Memory Primitives from Intel SGX," in *Proc. 2018 Netw. Distrib. Syst. Secur. Symp.*, Reston, VA: Internet Society, 2018, ISBN: 1-891562-49-5. DOI: 10.14722/ndss.2018.23239. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018{_}02B-4{_}Sasy{_}paper.pdf.