A Survey on SGX Enclave Privileged Side-Channel Attacks

AMIN FALLAHI, Syracuse University, USA

Side-channel attacks that target the programs executed on Intel SGX processors are of common vulnerabilities in this platform. The security of code and data in the cloud is provided by execution of the program in the SGX enclave, but it is subject to side-channel attacks which target the enclave and retrieve data by observing page-faults, cache-misses, TLB misses, etc. In this report, we review and discuss common attacks on SGX enclaves and the trending protection schemes discussed in current research works. In our study, we have observed and analyzed several types of research conducted in this area and we have compared them with each other.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Hardware-based security protocols**; **Side-channel analysis and countermeasures**; *Distributed systems security*; • **Computer systems organization** → *Processors and memory architectures*;

Additional Key Words and Phrases: Side-Channel Attack, Software Guard Extension, Page-Monitoring Attack

1 INTRODUCTION

One of the most recent challenges in cybersecurity systems is to outsource programs to be executed in the cloud maintaining the privacy of the owner and the security of data at the same time. The program code has to be protected alongside the data which is being used by it. Even in a securely encrypted code, the low-level instructions are visible when they are being executed by the CPU and a malicious operating system kernel can tamper with them. Several attacks are possible by reverse engineering the instructions, including code reuse, data injection, copyright violation, and recovery of classified data. Intel SGX (Software Guard eXtensions) [9] is a novel architecture introduced with Intel Skylake micro-architecture [12] that protects the code and the data inside an encrypted enclave that is a portion of memory protected by the support of hardware. The enclave is secure against direct eavesdropping by the adversary, but several successful attacking methods using side-channel information exist becuase this information are visible to the adversary [30]. The main drawback of this hardware extension is that the memory layout is still visible to the operating system kernel. Therefore, a malicious OS in the cloud can recover useful information by observing the occurrences of page faults. As a result, this side-channel attack can be successful on recovering the data by clearing the page table and causing page faults, then observing them for different inputs. In the same scenario, cache-misses are a point for side-channel attacks. The pattern of cache-miss occurrences can be used by a malicious operating system to attack the enclave program.

Several theoretical and practical attacks based on SGX side channels are discussed and experimented to be practical and effective. The points of interest during the execution of a program inside enclave for the adversary are cache and whatever is missed and retrieved from the memory, page-faults, and page-table.

In response to the development of attacks, several defense mechanisms are designed which have been effective in some cases and conditions. These defense methods usually sacrifice performance or accuracy while improving the security against side-channel attacks.

In this report, we study the attacks which are possible using SGX side-channel information and their points of success. Then, we discuss possible defense mechanisms which have been proposed by the researchers, study their methods and compare the defenses and their effectiveness under different conditions and scenarios.

In summary, the main points of study in this work are:

Author's address: Amin Fallahi, Syracuse University, 111 College Pl 4-294, Syracuse, NY, 13244, USA, afallahi@syr.edu.

- Examining SGX enclave points of attacks.
- Discussing major attacks which have proved to be successful against the enclave.
- Introducing the past work which have been done on defending the SGX enclave against side-channel attacks.
- Analyzing the defense methods and their effectiveness.

Outline In the remainder of this report, we discuss basic background knowledge required for studying this work in section 2. Then, we review and analyze the attacks which are possible to be launched against SGX enclave using side-channel information in section 3. In section 4, we introduce several defenses proposed by past research works, and in section 5 we analyze and compare the effectiveness of these defense methods in terms of security and performance. Finally, we conclude the research in section 6.

2 BACKGROUND

In this section, we provide detailed background on side-channel attacks targeting enclaved applications, tools, and terms we use in this report.

2.1 Storage Structure

In the common hardware architectures used in cloud computing environments today, data is transferred through several buses using multiple protocols to be used. The encrypted data is stored on the hard drive and will be transferred into enclaved memory. Then, it will be cached using multiple levels of cache for performance optimization. In current Intel processors, the cache hierarchy is made of three levels of fast storage. The last level cache (LLC) is shared among cores while each core has a separate pair of L1 and L2 caches. However, L1 and L2 are shared between the hyperthreads running on the same core. Furthermore, L1 caches data and instructions into different units while L2 and L3 share the space for both instruction and data [5]. Page mappings between virtual and physical addresses of data are also kept in the main memory and a transaction look-aside buffer (TLB) is used to cache the mappings in the page-table for better performance.

Each data transfer can be a point of side-channel attack. While the data is encrypted, the patterns of reads and writes, cache-misses, page-faults, and other interrupts and exceptions can be used by the adversary for launching attacks and recovering sensitive information.

2.2 Intel SGX

Protecting the data inside a shield has been a major interest for improving security and privacy in distributed systems. For this purpose, several software and hardware based solutions have been proposed.

Intel introduced a new hardware-based technique in its Skylake micro-architecture [12] line of processors called Software Guard eXtensions (SGX) [9]. It delivers 17 new instructions that can be used by applications to create enclaves, which are a portion of main memory assigned to a program and protected from being read and written by other processes. The effectiveness of this interface scales up to prevent the operating system kernel and hypervisor from accessing the shielded memory. The only trusted hardware component by SGX is the CPU itself and the plain text data can be stored in the caches and register while they are encrypted inside the main memory [5].

The enclave will be created by the operating system as a contiguous physical memory area and is a part of the virtual address space of an operating system process. The base address of the enclave will be assigned randomly from a dedicated region of physical memory called Enclave Page Cache

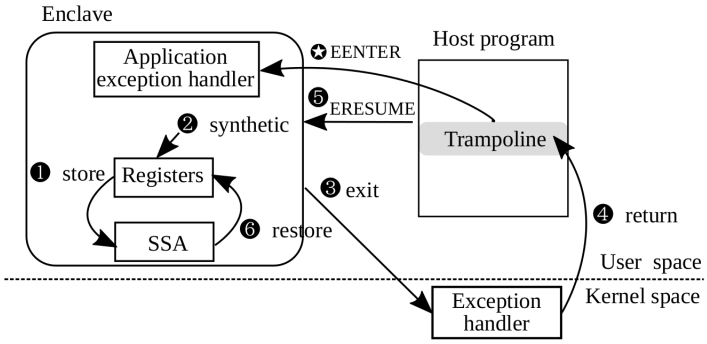

Fig. 1. Steps of an SGX asynchronous enclave exit [25].

(EPC) [5]. The security and confidentiality of the enclave is guaranteed by a hardware level memory encryption engine.

The virtual to physical address translation and transfering the data into enclave is also done by the OS. Meanwhile, SGX logs the behavior of the OS and records it for later observation by an external party for verification. Enclave binary is loaded by the operating system and since it is known by the OS, it cannot carry hard-coded sensitive data/code and all the secret information has to be loaded into the enclave during the runtime [23]. After creating the enclave and transferring the code/data into it, the OS will treat it as a normal process. Thus, it can be interrupted, migrated between cores, or halted. In the case of a context switch, SGX is responsible for cleaning the registers and saving the context. This process is called an asynchronous enclave exit (AEX). When the enclave program is to be executed again, the hardware will securely restore the context [5]. Detailed steps of an enclave exit can be described as below: [25] (Fig. 1)

(1) The processor stores register values and the exit reason into the state save area (SSA) inside the enclave.
(2) The processor loads synthetic data into registers.
(3) The enclave exits directly to the kernel space exception handler.
(4) The exception handler handles the interrupt and returns to the trampoline.
(5) The trampoline resumes the enclave.
(6) The processor restores the stored register values and resumes enclave execution. In addition, the trampoline can call an application exception handler inside the enclave to handle exceptions the OS cannot process.

Intel SGX SDK [10] which is used by developers to harden their applications using SGX technology, uses two main forms of specialized function calls to work with the enclave. ECALLs are functions which run inside the trusted enclave and can be called from outside untrusted environment while OCALLs are useful for calling non-sensitive system functions and other operations which are possible to be done outside the enclave [8].

It is also worth it to introduce two of the other Intel technologies which are dedicated to security systems in the following.

2.2.1 Trusted Platform Module. TPM is a security module, a microchip that is designed to enable some security measures and processes in the system. It is used to authenticate hardware devices using a unique secret key programmed in the chip. So, the security is pushed down to the hardware

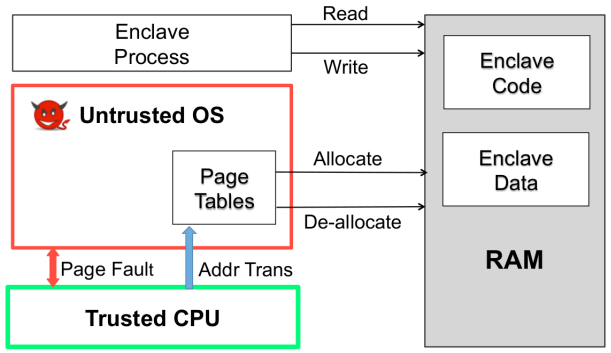

Fig. 2. Problem Setting. Process executing in an enclave on untrusted OS [26].

level. Basically, the TPM will identify if any firmware or boot process has been tampered with, while SGX protects sensitive and confidential processes executed by the operating system [2, 3].

2.2.2 Trusted Execution Technology. Intel TXT is a hardware technology designed to validate that no unauthorized manipulation has been done against the code that provides the secure environment at the time of launch. TXT uses the TPM chip and other cryptographic techniques to enable attestation of the authenticity of a platform and its operating system. On the other hand, SGX is designed to increase the security of application code [1].

2.3 Side-Channel Attacks on the Enclave

Multiple effective attacks and defenses on the enclave have been proposed by past research. Most of the attacks make use of the side-channel information that can be leaked and accessed by a modified malicious kernel. In these attacks, the malicious kernel retrieves useful information about the code and the data by changing the behavior of system, program execution or analyzing the page-faults and other possible points of data leakage.

One of the major points of attacks is targeting exceptions caused by the program. Since the exceptions are first acknowledged by the kernel before the program, a malicious OS can use them to infer activities and the behavior of the program according to the inputs which results in recovering the actual data and code of the program.

Another kind of attack that can be triggered by a malicious OS is an interrupt-based attack. The kernel can trigger interrupts to control the execution of the program.

Other attacks such as Stealthy Page Monitoring (SPM) attacks [29, 30] can also be effective. For a typical page-fault attack, the attacker can make all the page table entries invalid and restrict enclave program's access to all the pages and by causing page-faults, view the pattern of memory reads and writes by the program. This action will cause a lot of page-faults which lead to a high overhead and attack detection by the victim, so the attacker is interested to retrieve this information and view the pattern without causing unwanted page-faults. In page monitoring attacks, the attacker monitors the bits representing the validity and whether a page is accessed or modified. The attacker is also able to clear these bits and view them again frequently. So, the pattern of accesses can be visible to the attacker without causing page-faults [29, 30].

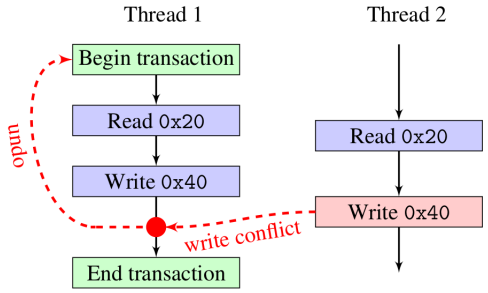

Fig. 3. RTM ensures that no concurrent modifications influence the transaction, either by preserving the old value or by aborting and reverting the transaction. [16].

2.4 Transactional Synchronization Extensions

One of the technologies that has attracted a great attention for making benefit in defense against SGX enclave side-channel attacks is Intel TSX [13]. This technology has been introduced by Intel since Haswell micro-architecture [7] for critical sections management and monitoring serialization. Intel TSX provides two frameworks to add transaction support to the processor. Both frameworks introduce new instructions for region specification, backward compatibility, and flexibility [13]. One of them is RTM (Restricted Transactional Memory) [11] which has been used by previous research like T-SGX [25] to isolate instructions and Cloak [16] to pin sensitive code and data to the caches which are not supported by current hardware instructions. Both of these works make use of transactions to protect the enclave against side-channel attacks.

TSX introduces three main instructions XBEGIN, XEND, and XABORT. The instruction code will reside between XBEGIN and XEND while XABORT will cause the transaction to abort. Upon abortion, a fallback function will be called to handle reversing the transaction effects (Fig. 3).

These transactions can be used to protect and isolate code. While unexpected faults can cause a transaction to exit, the SGX enclave can make benefit of it to protect the code/data against faults.

3 ATTACKS

In this section, we introduce and discuss the attacks which use side-channel information in a SGX environment. We will discuss major points of attack that have been studied by the past research and then describe one attack related to each. At the end we discuss some other attacks in summary.

3.1 Attack Model

In our attack scenario, the enclave is protected by Intel SGX, so no one except the main program running inside the enclave has the key to access it. The operating system kernel which runs on top of a hypervisor can view and manage tasks queue, exceptions and interrupts. The hypervisor kernel is also capable of viewing data transfer between memories and caches while Intel SGX prevents it from accessing the enclave data. It is considered that other tenants executing on the same hypervisor are not able to control the resources assigned to the main machine by the hypervisor.

3.2 Points of Attack

Several caveats with SGX architecture have been studied to be points for launching attacks against the enclave.

ALGORITHM 1: Prime+Probe branching side-channel sample code [16, 29]

```
if (secret)
{
    ;
}
else
{
    ;
}
```

TLB, as an address translation cache, can be one point of attack. While TLB is shared between enclave and non-enclave programs when hyperthreading is enabled, it can create side-channels which can be used for retrieving useful information [30].

Also, during AEX (Asynchronous EXit), TLB and paging-structure caches will be flushed and let the adversary know the flushed entries at context switch [30].

Other than TLBs, CPU caches which are shared between the enclave and non-enclave code can result cache side-channel attacks [30].

Another point of attack is the use of page table entry flags. The accessed and dirty bits which are set and unset for page table entries upon accesses and modifications are used for launching attacks. So, the code in non-enclave mode can observe page-table updates and the memory write pattern to recover useful data [29, 30].

3.3 Cache-Based Attacks

Attacks which use cache for eavesdropping, usually work on causing intentional cache misses. Since the cache is not visible to the OS, the attacker is interested to encourage the program to access the main memory.

Attacker uses parameters like time, trace, and access patterns to infer sensitive data from the cache. When the attacker shares the same direct-mapped cache set with the victim, it is possible to launch a Prime+Probe attack against the victim. Considering an L1 cache of 64 sets, each containing 8 cache lines which are each 64 bytes, in the prime stage, the attacker executes conflicting cache lines to make cache misses possible for the victim who branches on a secret (Alg. 1). By executing virtual addresses that map to the same cache set using a sequential memory scan for an address pair, the attacker expects the victim to request access to a conflicting cache line and cause a cache miss [23]. In the probe step, the attacker executes all the cache lines and measures the execution time of each, and based on it, the attacker can find out if that cache line has been touched by the user, because the execution on the particular cache line will take more time due to cache miss. So, if the victim is continuously branching on a secret, it will be revealed which branch has been taken. Based on different inputs, the attacker can guess the behavior of the program [16, 20, 28] (Fig. 4, 5).

Most of the past researches work on extracting RSA and EdDSA keys as the most used cryptography systems [5, 16, 23, 26, 29, 30]. However, both follow the same path and basically RSA is expandable to EdDSA and both are vulnerable to prime+probe attack. As shown in Alg. 2, while RSA is based on calculating an exponentiation modulo some number n, using a square+multiply algorithm it is possible to calculate a result in a branch based algorithm. The calculated result inside the loop is dependent on the bits in d which is a secret than can lead the attacker to recover the private key. In each iteration, if the bit is one, the result will be calculated based on a square and a multiply, otherwise it will be calculated based on a single multiplication. Using this branch, it is possible to find out if the multiply is executed or not and then recover the bitstream of the key.

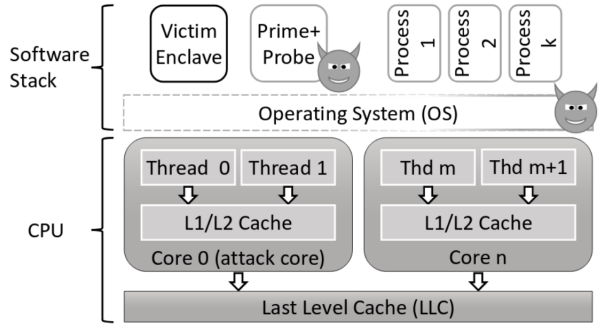

Fig. 4. A high-level view of the Prime+Probe attack; victim and attackerâĂŹs Prime+Probe code run in parallel on a dedicated core. The malicious OS ensures that no other code shares that core to minimize noise [5].

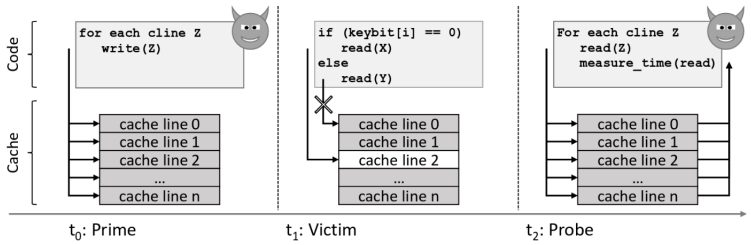

Fig. 5. A detailed view of Prime+Probe side-channel attack; first the attacker primes the cache, next the victim executes and occupies some of the cache, then the attacker identifies which cache lines have been used by the victim. This information allows the attacker to recover secret data in use by the victim process [5].

ALGORITHM 2: Square and multiply exponentiation [23]

input: base b, exponent e, modulus n
output: $b^e \, mod \, n$
$X \leftarrow 1$;
for $i \leftarrow bitlen(e)$ **downto** 0 **do**
    $X \leftarrow multiply(X, X)$;
    if $e_i = 1$ **then** $X \leftarrow multiply(X, b)$;
    end
end
return X;

Thus, this secret dependent memory access leads to time difference between the cached data and uncached data which is used by the prime+probe attack [16, 23].

While the private key is kept inside an SGX enclave, it makes sense to also do the attack inside an SGX enclave to leave the minimum possible trace for the detection of the attack by the victim [23].

A common problem that comes into view in terms of making the attack more practical, is that Intel has disabled its high precision timer (rdtsc) instruction to be used inside the enclave for security reasons. So, it is needed to implement a high precision timer inside an enclave which has been done by [23] as shown in Alg. 3. They start a thread for continuously incrementing a global

ALGORITHM 3: Timestamp counter [23]

```
mov &counter, %rcx
l: inc %rax
mov %rax, (%rcx)
jmp l
```

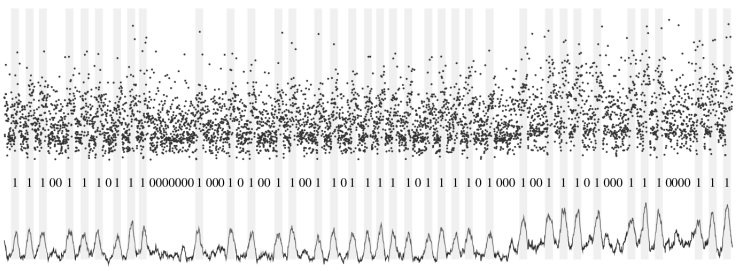

Fig. 6. Measurement's raw trace and over moving average. It is visible that the secret bits can be identified [23].

variable and achieve even better performance than Intel's high precision timer. The number of cycles which are executed inside the loop are constant and a parameter of time, enough for precise specification of a relative time scale [23].

Another problem is to find an eviction set to load into cache for causing row conflicts. For this purpose, since the enclave memory is allocated in a contiguous way, it is possible to sequentially scan the memory for conflicting pairs of addresses.

A prime+probe attack based on this setting, can result in a pattern observation as shown in Fig. 6. By running a moving average on the result, it is possible to extract the key more easily [23].

All has been done is experimented to be perfectly hidden, only some system noise can cause instability which is almost undetectable in comparison to an attack outside the enclave [23].

3.4 Page Table Based Attacks

Several studies have been done on attacks which make use of page tables and page faults to extract sensitive data from the enclave. A basic attack is to make use of AEXs which have been caused when a page fault occurs and results in the base address of the faulting page to be revealed. A malicious operating system can use this fact to cause intentional page faults by clearing the present bits in page table or evicting the page table entries. Then, the OS can obtain the page level trace of the execution [29]. A similar attack is to monitor a specified page and measure the time between page faults that occurred for that page. Every time a page fault occurred, the OS invalidates the data for the next access and supposedly page fault.

As an example, for analyzing genome data inside an enclave as shown in Fig. 7 and reporting the presence of specific mutations, the page containing *add_description_xyz* method is evicted from EPC memory. When the page fault is raised during the execution, the method is called and patient's specific disorder can be inferred by the attacker [27].

Similarly, the attack can be launched against both code and data. In Fig. 8(a) the operating system can find out the gender of the user if it makes each function to be on different code pages. In Fig. 8(b), by putting the function *CountLogin* and variables *gMaleCount* and *gFemaleCount* in different memory pages, the attacker can first find out if the function has been called by observing the page fault for the page containing *CountLogin* function. Then, he can find out which variable has been

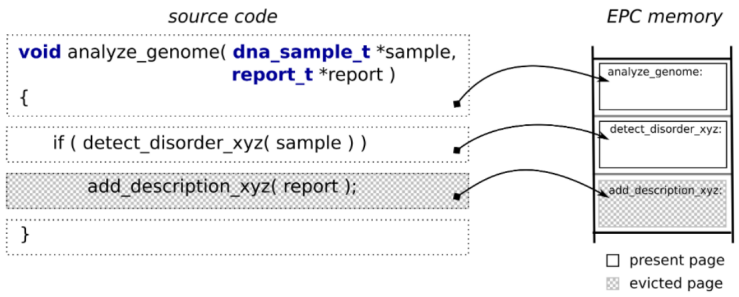

Fig. 7. Basic example of a page-table based side channel attack. The occurrence (or absence) of a page fault after an attacker evicted the *add_description_xyz* method reveals the result of the DNA analysis [27].

incremented by observing the two pages containing *gMaleCount* and *gFemaleCount*. Gradually, the attacker can hold the number of males and females.

```
char* WelcomeMessage( GENDER s ) {
    char *mesg;

    // GENDER is an enum of MALE and FEMALE
    if ( s == MALE ) {
        mesg = WelcomeMessageForMale();
    } else { // FEMALE
        mesg = WelcomeMessageForFemale();
    }
    return mesg;
}
```

(a) Example function with input-dependent control transfer [27].

```
void CountLogin( GENDER s ) {
    if ( s == MALE ) {
        gMaleCount ++;
    } else {
        gFemaleCount ++;
    }
}
```

(b) Example function with input-dependent data access [27].

Fig. 8

The problem with the discussed attacks is that they induce a high overhead by causing lots of page faults. This performance issue will lead to the attacks being exposed by the victim. So, it is important to come out with a method for retrieving information without causing page-faults.

Van Bulck et al. [29] have confirmed that accessed and dirty flags are set and unset for page table entries in both enclave and non-enclave mode. Monitoring these flags can lead to page-fault detection on protected memory region without the need to produce intended page-faults. Suppose a program is being executed in the enclave which branches based on a secret. If the branch is taken, one page will be retrieved and thus its accessed flag will be set, otherwise the same happens for

another page. By observing the accessed bit for these two page tables, the secret is revealed. Doing this sequentially can reveal all the secrets [29].

A problem with this attack is that after address translations are cached in the TLB, the flags are no longer updated in the page-table, but are updated in the TLB. Thus, it is needed to force the victim to walk its page-tables in order for the flags to be updated. In this attack, the attacker flushes the TLB by initiating an inter-processor interrupt which causes a TLB shootdown. Thus, the flags will need to be updated in the page-table before being cached in the TLB [29, 30].

As a result, the high number of accesses to a page-table entry for a certain input, can result in lots of interrupt requests which can be detected by the victim. To prevent this, by finding out two pages which are accessed by the victim repetitively and measuring the time between these two page accesses, the input can be recovered. Thus, instead of multiple interrupts, one interrupt after each cycle is enough for clearing the TLB and successful attack [29, 30].

To further remove the interrupts which are needed by the two aforementioned attacks, it is possible to make use of the fact that the TLB is shared between two threads which are being executed on the same core. Thus, by running the attacker program in the same core with the victim program, it is possible to invalidate TLB entries outside the enclave without interrupt requests [29, 30].

3.5 Other Attacks

After discussing major cache and page table based attacks, we briefly introduce some other techniques which have been proposed by past research and make use of these two side channels to launch attacks against the enclave.

3.5.1 Flush+Reload. By making use of page sharing between enclave and non-enclave code, the attacker can find out if a certain page is removed from the cache. More precisely, the attacker flushes the target memory line from the cache and waits for the victim to access it. The wait time can be fixed to either a random value or a value that is experimented to be good for the attack success. Then, the attacker requests to access the page and measures the time it takes for the page to be fetched. Judging based on the time, the attacker can say if the page is already cached or fetched from the memory. In the case that it takes longer than a certain amount of time for the page to be retrieved, the attacker can conclude that it has not been fetched or touched by the victim [31].

3.5.2 Flush+Flush. Using the *clflush* instruction, it is possible to flush a cache line and if the cache line is already empty, the instruction will abort. Considering that the abort takes less time than the flush, it is possible to use this delay to recover information about the content of the cache. By observing the existence of data inside the cache line, it is possible to find out which memory address has been accessed by the victim. By running the *clflush* instruction repetitively and measuring its execution time, the attacker retrieves this information [17].

3.5.3 DRAMA. With caching disabled to prevent cache-based side-channel attacks, the attacker allocates two memory lines in the same memory segment which map to different virtual addresses but have common physical addresses. Then the attacker starts to regularly access one of the memory lines. In case the victim accesses the other memory line, a row conflict will cause the next fetch of the attacker to take more time. By measuring these times the attacker can recover the sensitive memory accesses by the victim [21, 30].

3.5.4 Cache-DRAM. Using two threads for launching the Prime+Probe attack [20, 28] and DRAMA attacks [21] together, Prime+Probe can cause conflicts in the cache which leads to fetching the data from DRAM, making DRAMA attack possible in the case caching is disabled. Therefore, by launching this attack, the slowdown caused by disabling cache in DRAMA attacks is avoided [30].

ALGORITHM 4: Cloak sample code [16]

```
xbegin();
preload all sensitive data/code;
run algorithm;
xend();
```

4 DEFENSES

We discuss the research that has been done on protecting the enclave against the mentioned side-channel attacks and their effectiveness. Some research study on preventing the attack from being performed and some intend to defend the enclave against attacks and block them.

4.1 Defense Against Cache Based Attacks

As discussed in 2.1, there are three levels of cache: L3 is shared between all physical cores, L2 is shared between each core's threads and L1 shares data and code separately between the hyperthreads. In a typical environment, it is not visible to the enclave program running in one thread, what is being run in the other thread on the same core. So, if the other hyperthread is trusted (assuming there are two hyperthreads running in each core) it is only needed to care about the L3 cache. In the SGX environment, the hyperthreads cannot trust each other, thus, L2 and L1 caches are also of importance when it comes to the case of cache side-channel attacks because attacks like Prime+Probe will be possible in this case [16].

The basic approach to confront cache-based side-channel attacks including Prime+Probe and eliminate the leakage is to pin sensitive code and data into caches so no conflict and cache-miss be possible, but it is impossible to pin data into caches in the current hardware/software architecture. So, there is a need to introduce a hack for indirectly doing this which has been implemented using Intel TSX by Gruss et al. [16].

By studying the in-depth architecture of TSX instructions, it is found that TSX uses a write set and a read set and interrupts to these sets cause the transaction to abort. The write set is always the 32kb L1 and the read set is the 8mb sized L3. So, if a cache line in L1 is modified and made dirty, the transaction aborts. The same happens when some cache line in L3 is removed due to a conflict. Also, it is observed that code, which is read-only, can only be a part of the read set L1. This has the benefit of detecting external code evictions by other cores which share the same L3 cache [16].

After explaining the indirect effect of using TSX for pinning data to cache, it is needed to load the data and code into the cache inside a TSX transaction. After that, each cache conflict which leads to making cache access pattern to be visible will cause the transaction to abort. So, the sensitive code and data will be preloaded and the program will start to run inside the transaction. Then, the victim does not experience cache misses and there is nothing for the attacker to measure, so the enclave is protected against the malicious OS [16].

For preloading, if the attacker targets L1, the code should be preloaded into L1-I by execution. Also, all the data should be loaded into the write set. For the attackers targeting L3, code and read-only data should be loaded into read set, while the data which will possibly be touched again should be loaded into the write set [16].

4.2 Defense Against Page Table Based Attacks

One of the late defenses proposed against page fault attacks is T-SGX [25] which has been developed by Shih et al. Based on the observation that race conditions cause Intel TSX transactions to abort, T-SGX puts the whole enclave inside a transaction. After abort, it will roll back all the changes and

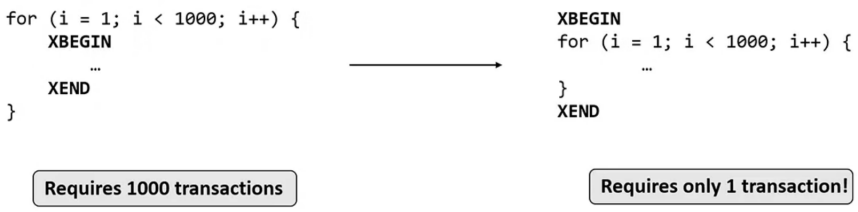

Fig. 9. An example of loop optimization [27].

control will be transfered to the fallback point. This way, any page fault can make the transaction abort and the enclave can self detect itself against it.

However, TSX is not designed to contain and execute huge amounts of code. A frequent timer interrupt by the OS can cause the transaction to abort. So, if the transaction takes too long to execute, it keeps aborting and the program will never terminate. Another constraint is that TSX buffers all the memory changes inside the cache. So, if a single transaction does lots of memory accesses a cache conflict will occur and the transaction will abort. To solve this problem, this research breaks the program into several tiny execution blocks. This method can eliminate the probability of unnecessary abortions.

A problem that comes with the introduction of tiny execution blocks is that setting up a transaction has a fixed cost in terms of number of processor cycles. This will have a bad effect on performance. So, any time it is possible and constraints are not violated, it is preferred to merge the continuous blocks like loops, if-else statements, and functions. For example, as shown in Fig. 9 one possible loop optimization technique is to move the transaction outside the loop to prevent high number of transaction creations. By static analysis, it is indicated if it is safe to optimize based on the constraints.

Another problem is that TSX instructions are still outside transactions and transaction boundaries are still vulnerable to page fault attacks in this design. The attacker can unmap the pages where blocks start by writing to them. This can be solved by putting all transaction begin and end pairs in a dedicated springboard page. This way, only one page will be leaked [25].

4.3 Other Defense Methods

Chen et al. [6] embed a clock inside the enclave and use Intel TSX for detecting any interrupt to it. When the execution time of the program inside the enclave exceeds a fixed amount of time as the threshold, it is considered that the code has been interrupted by some external factor and an AEX (asynchronous enclave exit) will be detected. The implemented clock should be hidden from the OS in terms of reading and modification and it should not go backward in time under any condition. These can be assured by implementation inside SGX enclave. Also, the clock should not be stopped or interrupted by the OS which is solved by using TSX. For protecting the clock to be read by the OS, a random value will be generated in a transaction and will be added to the global timer variable. This will run in a thread parallel to the main program inside the enclave (Fig. 10). The threshold delay needs to be approximated by training the system with AEX-free execution time and minimum delay of an AEX execution.

Shinde et al. [26] conduct a statistical study on the percent of leakage, vulnerability, possibility and effectiveness of page-fault based attacks which they call pigeonhole attacks. Then, they consider the possible ways of eliminating page-fault side-channels. More precisely, they define pigeonhole attack as an attack that is performed when a page table walk is done. Moving data between physical

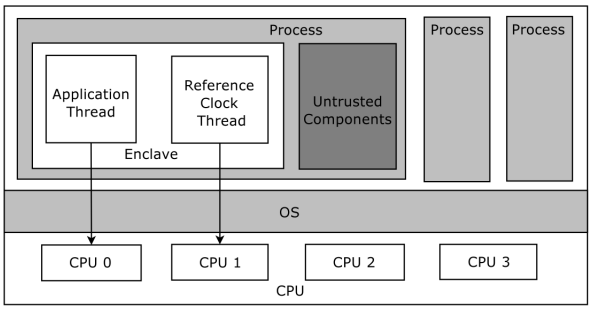

Fig. 10. System architecture of DÃľjÃă Vu. Blocks in gray are untrusted, which include the untrusted components of the processes and the entire OS kernel [6].

and virtual memory from different pages sequentially causes page-faults on every single page when the virtual memory is already full, holding all the other pages. When the first page is fetched, the next page will be removed from the page table and when the next page is to be fetched, a page-fault will occur. They create a deterministic page access and execution profile by introducing fake accesses so the program will access the same pattern of the pages independent of the input and make it page-fault oblivious. The page containing the input dependent data is called data staging page and the one containing the code is called code staging page. By creating a decision tree of the program instructions, it is understood that in each stage of the program which is being executed, there is a page-fault sequence dependent on the data, which can be mapped to a staging page. So, for each stage, one staging page is considered. This technique will create a high overhead, thus, the research extends to optimizing the implemented method by eliminating copy operations which is done by skipping read-only data save, merging small data blocks into single pages to reduce the number of fetches, merging levels of execution and reducing the number of stages, removing the mux based on the input to reduce the possibility of branch conditions, and removing the dependence of code on the data (Ex. CAS obliviousness).

Seo et al. [24] study address space layout randomization (ASLR) which is used in traditional systems for defending the memory against corruption attacks, but SGX does not make benefit of it. They present a new ASLR scheme for SGX systems and study the new challenges which come with it. Including the problem of small randomization entropy which is a result of limited physical and enclave memory and makes brute force attacks possible. Also, the inability of runtime page permission change makes the enclave vulnerable to code injection attacks due to conflicts with ASLR which is addressed by implementing a software-based RWX permission set.

Sasy et al. [22] implement a block level memory controller as a new library combining ORAM [14, 15, 19] and Intel SGX while mitigating the disadvantages of these technologies and protects the enclave from software-based side-channel attacks meanwhile optimizing the performance.

Brasser et al. [4] propose data location randomization as a new approach for defending enclave side-channel attacks by designing and implementing a compiler based tool to shuffle the in-enclave data locations at the granularity of cache lines.

5 ANALYSIS

Cloak [16] has achieved significant protection against cache-based side-channel attacks. However, there will still be some leakage remaining which this research does not handle. Aborts do not cause the concurrent memory accesses to be canceled. Branch predictor is influenced due to the cache pinning and other micro-architecture effects can be possible due to the modifications to the

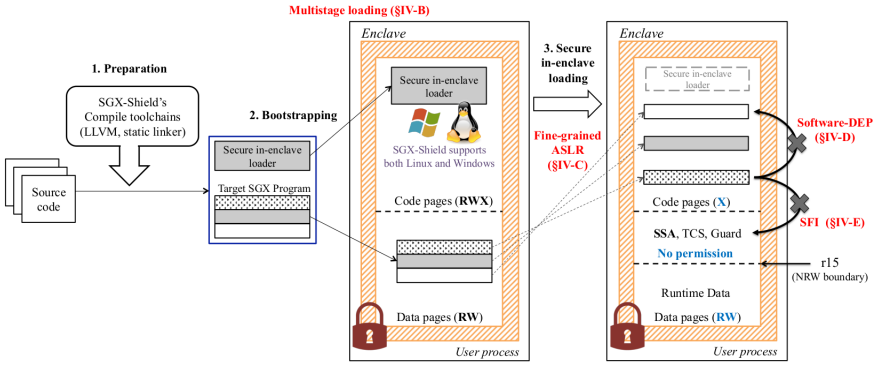

Fig. 11. Overall workflow of SGX-Shield: 1) the preparation phase builds a SGX binary from the target programâĂŹs source code; 2) the bootstrapping phase loads the secure in-enclave loader into code pages and the target SGX program into data pages; and 3) the secure in-enclave loading phase finally loads the target SGX program [24].

execution behavior. On the other hand, the introduction of a new software-based way for pinning data into cache lines and making use of transactions for this purpose is a novel idea that opens new ways for future research in several areas. There is also an opportunity for later research on optimization and performance improvement and architectural study for reducing faults.

T-SGX [25] protects the enclave against page-fault based side-channel attacks, but at the same time, attacks which are based on monitoring accessed and dirty bits of the data are not detected by this method. Also, this attack incurs 30% memory overhead and 50% runtime overhead which has a significant effect on the performance.

The problem with Deja Vu [6] is that it only considers enclave exits as points of attacks and possible attacks that may have been occurred during the execution which do not cause AEX have been excluded. So, page monitoring attacks which do not rely on AEX can still be effective [30].

Overall, all the techniques based on using transactions to isolate the program from interrupts and immune them to page-faults, need an isolated executing environment which causes a single thread execution, low utilization, and high overhead.

Randomization, as studied in [24] can be a leading point for research on possible defenses. However, it is used to implement an ASLR for SGX which is not lively randomized. Thus, the attacker can observe the random patterns of memory accesses for multiple times and infer an approximate access pattern which is helpful for launching attacks [30]. Also, the implementation is not evaluated against brute force attacks, so, the amount of introduced entropy cannot be verified to be effective.

The work done by Shinde et al. [26] does not take cache-based side-channel attacks into account. The idea of loading all the sensitive data into the same pages can still leak information using cache and TLB side channels [30]. Also, fake accesses can cause overhead and instability.

The bottom line is that none of the proposed methods can completely defend the enclave against the possible side-channel attacks. Page-fault detection cannot protect the enclave since page accesses can be learned without causing page faults. It is not even needed to interrupt the enclave for reading the TLB. Thus, a solid software based defense method needs to either remove all the branches and conditional code/data from the program or find a way to reliably pin specific data/pages to the cache and therefore TLB.

Table 1. Effectiveness of proposed defenses against attacks

Attack/Defense	P+P	Basic PF	F+R	F+F	DRAMA	PM
Cloak	✓	✗	✓	✓	✗	✗
T-SGX	✗	✓	✗	✗	✓	✗
Deja Vu	✗	✓	✗	✗	✓	✗
Shinde et al.	✗	✓	✗	✗	✓	✓
SGX-Shield	✗	✓	✗	✗	✓	✗
ZeroTrace	✓	✓	✓	✓	✓	✓
DR. SGX	✓	✗	✓	✓	✗	✗

A hardware based solution can be combining Cache Allocation Technology (CAT) with SGX. CAT provides a software to control where data is allocated into the LLC [**?**]. This can make sure that some cache sets can only be used by one CPU core so there will be no shared cache set to launch cache based attacks. A better approach can be providing a non-shared secure memory element which is not cached [23].

Finally, we aggregate the effectiveness of the discussed defenses against the attacks we introduced in section 3 in Table 1. As we can see, most of the defenses either work on cache based attacks or page table based attacks or combine both. A reliable single solution to defend both kinds of attacks is still subject to research. The Page Monitoring (PM) attack can defeat most of the defense approaches and as a newly introduced attack is subject to more study. ORAM based defenses are also interesting for further research and can achieve good protection if there is a way to improve performance.

6 CONCLUSION

In this report, we studied various kinds of attacks proposed by the past research and also reviewed the defenses in terms of their technique, security, and performance, also analyzed their effectiveness. While several attacks can be practical to a good extent, there are still points of attacks and there is still space for security and performance improvements. Finally, the area is subject to further extensive research according to various unknown attacks which emerge every day.

ACKNOWLEDGMENTS

This work has been done for the fulfillment of Ph.D. candidacy as qualification examination 2 (QE-2) at Syracuse University Department of Engineering and Computer Science for Computer and Information Science and Technology (CISE) program.

REFERENCES

[1] [n. d.]. Intel SGX vs TXT. https://intelsgx.blogspot.com/2016/05/intel-sgx-vs-txt.html

[2] [n. d.]. What are the functional similarity and difference between TPM and SGX in trust computing? https://security.stackexchange.com/questions/175749/what-are-the-functional-similarity-and-difference-between-tpm-and-sgx-in-trust-c

[3] 2018. Trusted Platform Module. https://en.wikipedia.org/wiki/Trusted_Platform_Module

[4] Ferdinand Brasser, Srdjan Capkun, Alexandra Dmitrienko, Tommaso Frassetto, Kari Kostiainen, Urs Müller, and Ahmad-Reza Sadeghi. 2017. DR. SGX: Hardening SGX Enclaves against Cache Attacks with Data Location Randomization. *arXiv preprint arXiv:1709.09917* (2017).

[5] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure: SGX cache attacks are practical. *arXiv preprint arXiv:1702.07521* (2017), 33.

[6] Sanchuan Chen, Xiaokuan Zhang, Michael K Reiter, and Yinqian Zhang. 2017. Detecting privileged side-channel attacks in shielded execution with Déjá Vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and*

Communications Security. ACM, 7–18.

[7] IntelÂő Corporation. [n. d.]. 4th Generation IntelÂő CoreâĎć Processor. Retrieved April 12, 2018 from https://software.intel.com/en-us/node/524022

[8] IntelÂő Corporation. [n. d.]. ECALL/OCALL Functions. Retrieved April 12, 2018 from https://software.intel.com/en-us/node/709001

[9] IntelÂő Corporation. [n. d.]. IntelÂő Software Guard Extensions (IntelÂő SGX). Retrieved April 12, 2018 from https://software.intel.com/en-us/sgx

[10] IntelÂő Corporation. [n. d.]. IntelÂő Software Guard Extensions (IntelÂő SGX) SDK. Retrieved April 12, 2018 from https://software.intel.com/en-us/sgx-sdk

[11] IntelÂő Corporation. [n. d.]. Restricted Transactional Memory Overview. Retrieved April 12, 2018 from https://software.intel.com/en-us/node/524022

[12] IntelÂő Corporation. 2017. 6th Generation IntelÂő Processor Family. Retrieved April 12, 2018 from https://www.intel.com/content/www/us/en/processors/core/desktop-6th-gen-core-family-spec-update.html

[13] IntelÂő Corporation. 2017. IntelÂő Transactional Synchronization Extensions (IntelÂő TSX) Overview. Retrieved April 12, 2018 from https://software.intel.com/en-us/node/524022

[14] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 182–194.

[15] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473.

[16] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and efficient cache side-channel protection using hardware transactional memory. In *USENIX Security Symposium*.

[17] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.

[18] Mehmet Kayaalp, Nael Abu-Ghazaleh, Dmitry Ponomarev, and Aamer Jaleel. 2016. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 72.

[19] Rafail Ostrovsky. 1990. Efficient computation on oblivious RAMs. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM, 514–523.

[20] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *CryptographersâĂŹ Track at the RSA Conference*. Springer, 1–20.

[21] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks.. In *USENIX Security Symposium*. 565–581.

[22] Sajin Sasy, Sergey Gorbunov, and Christopher Fletcher. 2017. ZeroTrace: Oblivious memory primitives from Intel SGX. In *Symposium on Network and Distributed System Security (NDSS)*.

[23] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[24] Jaebaek Seo, Byounyoung Lee, Seongmin Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. 2017. SGX-Shield: Enabling address space layout randomization for SGX programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*.

[25] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*.

[26] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2015. Preventing your faults from telling your secrets: Defenses against pigeonhole attacks. *arXiv preprint arXiv:1506.04832* (2015).

[27] Raoul Strackx and Frank Piessens. 2017. The Heisenberg Defense: Proactively Defending SGX Enclaves against Page-Table-Based Side-Channel Attacks. *arXiv preprint arXiv:1712.08519* (2017).

[28] Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology* 23, 1 (2010), 37–71.

[29] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *Proceedings of the 26th USENIX Security Symposium*. USENIX Association.

[30] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, Xiaofeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A. Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. 2421–2434.

[31] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack.. In *USENIX Security Symposium*. 719–732.