A Survey on SGX Enclave Privileged Side-Channel Attacks

AMIN FALLAHI, Syracuse University, USA

Side-channel attacks targeting programs which are 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 protection which is needed and currently trending in research. In our study, we have observed and analyzed several types of research done in this area and 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 present challenges in cybersecurity systems is to outsource programs to be executed in the cloud and at the same time maintaining the privacy of the owner and the security of data. 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) [6] is a novel architecture introduced with Intel Skylake micro-architecture [9] that protects the code and the data inside an encrypted enclave that is a portion of memory which is protected by the support of hardware. The enclave is secure against direct eavesdropping by the adversary, but there are several successful attacking methods using side-channel information which are visible to the adversary [26]. The problem with this hardware extension is that the memory layout is still visible to the operating system kernel and 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 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.

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

In this report, we study the possible attacks which are possible using SGX side-channel information and their points of success. Then, we discuss possible defenses 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:

- Examining SGX enclave points of attacks.
- Discussing 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 which have been proposed by past research and in section 5 we analyze and compare the effectiveness of these defense methods in terms of security and performance. At last, 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 Intel SGX

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

Intel introduced a new hardware-based technique in its new Skylake micro-architecture [9] line of processors called Software Guard eXtensions (SGX) [6]. The main function of it is assigning a portion of main memory to a program and protecting it from being read and written by other processes. The effectiveness of this technique scales up to prevent the operating system kernel itself from accessing the shielded memory which is called enclave. Intel SGX API [7] uses two main kinds of specialized function calls to work with the enclave. ECALLs are functions which run inside the enclave and can be called from outside and OCALLs are useful for using system functions and other operations which are possible to be done outside enclave [5].

One of the major problems with running applications inside the cloud is the leakage of data and code instructions. Even with encryption, the CPU can only execute unencrypted instructions which can be eavesdropped and reverse engineered to useful code. Only with hardware modifications and adding SGX support to the hardware, it is possible to improve security and privacy in this term.

2.2 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. Also, page mappings between virtual and physical addresses of data are 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.

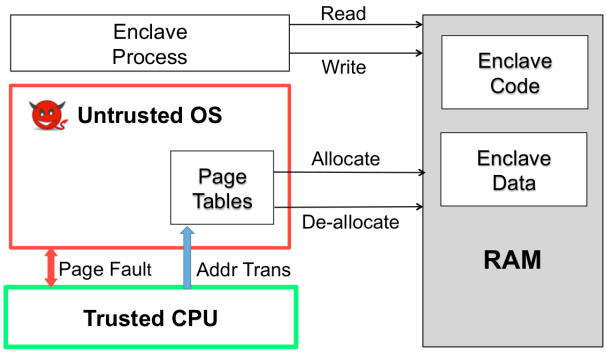

Fig. 1. Problem Setting. Process executing in an enclave on untrusted OS [23].

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 useful information.

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 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 [25, 26] 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 by the victim. 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 [25, 26].

2.4 Intel TSX

One of the technologies which have came into notice for making benefit in defense against side-channel attacks is Intel TSX [10]. This technology has been introduced by Intel since Haswell micro-architecture [4] 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,

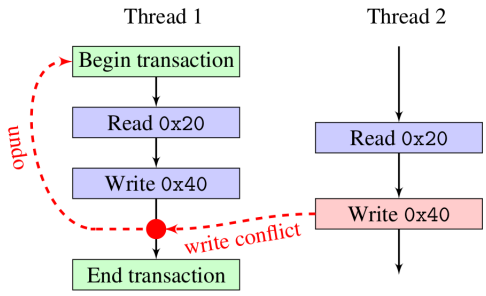

Fig. 2. HTM ensures that no concurrent modifications influence the transaction, either by preserving the old value or by aborting and reverting the transaction. [13].

and flexibility [10]. One of them is RTM (Restricted Transactional Memory) [8] which has been used by previous research like T-SGX [22] to isolate instructions and Cloak [13] to pin sensitive code and data to the caches which is not supported by current hardware instructions.

3 ATTACKS

We introduce and discuss the attacks which use side-channel information in this section and focus on summarizing the methods and points of attack.

3.1 Attack Model

In our attack scenario, the enclave is protected by Intel SGX, so no one except the main program running inside 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 weaknesses in SGX architecture have been studied to be points for launching attacks against the enclave.

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 [26].

Also, during AEX (Asynchronous EXit), TLB and paging-structure caches will be flushed and let the adversary know the flushed entries at context switch [26].

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 [25, 26].

Other than TLBs, CPU caches are shared between the enclave and non-enclave code resulting in the possibility of cache side-channel attacks [26].

ALGORITHM 1: Prime+Probe branching side-channel sample code [13, 25]

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

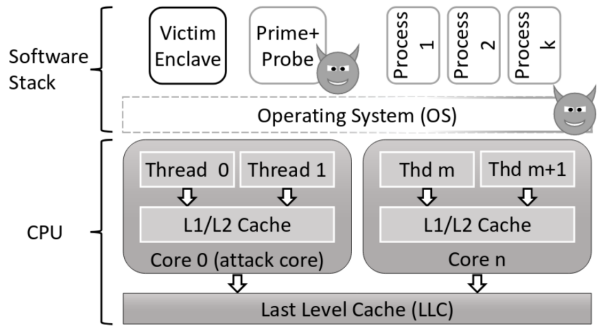

Fig. 3. A high-level view of the Prime+Probe attack; victim and attacker's Prime+Probe code run in parallel on a dedicated core. The malicious OS ensures that no other code shares that core to minimize noise [2].

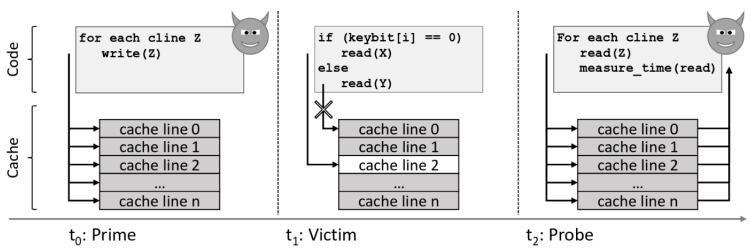

Fig. 4. A detailed view of Prime+Probe side-channel attack; first the attacker primes the cache, next the victim executes and occupies some of the cache, then the attacker identifies which cache lines have been used by the victim. This information allows the attacker to recover secret data in use by the victim process [2].

3.3 Prime+Probe

Prime+Probe is composed of two stages. 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 page fault possible for the victim who branches on a secret (Alg. 1). In the probe step, the attacker executes all the cache lines and measures the execution time of each, and if a cache line execution takes longer, it can be revealed that the secret has been zero [13, 18, 24] (Fig. 3, 4).

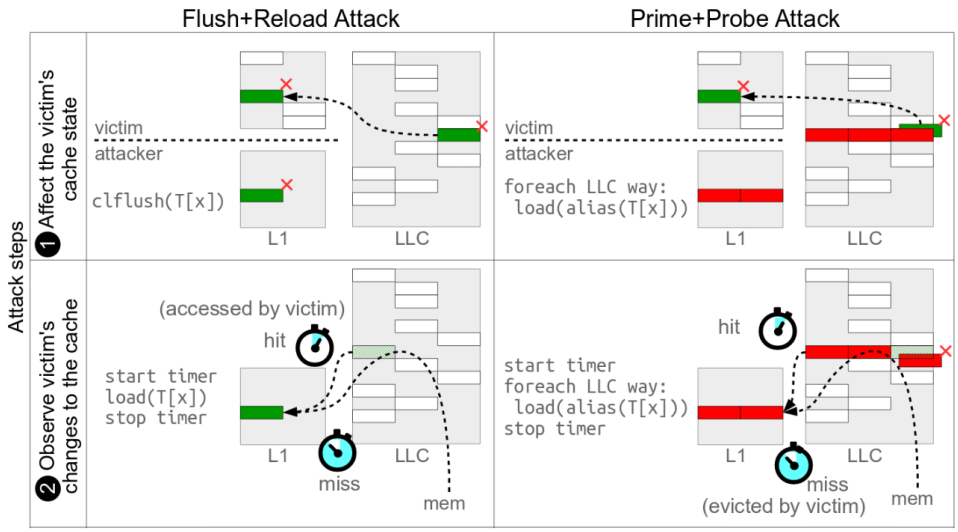

Fig. 5. Prime+Probe vs Flush+Reload Attacks [15].

3.4 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 [27].

3.5 Stealthy Page Table-Based Attacks

Since the emerging of various page-fault based side-channel attacks, it is important to come out with a method for retrieving information without causing page-faults. Intended page-faults are possible to protect, and also cause a lot of overhead which leads to attack detection.

Van Bulck et al. [25] have confirmed that accessed and dirty flags are set and unset for page table entries in both enclave mode and non-enclave mode. Monitoring these attacks 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 occurs 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 which lead to EdDSA session key to be revealed gradually [25].

3.6 Sneaky Page-Monitoring Attacks

Wang et al. [26] introduce three kinds of attacks for monitoring page-table entries with the consideration of the performance slowdown caused by producing intentional page-faults. In the case that a page-fault occurs for each memory access by the victim, the number of page-faults and performance slowdown can be a point for attack detection.

3.6.1 Accessed flags monitoring. Accessed and dirty flags of page table entries, as discussed earlier, can be used to trigger attacks without causing page-faults. In this attack, the attacker monitors the set-unset pattern of the flags for page table entries and recovers information about the program running inside the enclave and encrypted data which is being accessed in the protected memory. 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 TLB shootdown. Thus, the flags will need to be updated in the page-table before being cached in the TLB [25, 26].

3.6.2 Timing enhancement. The high number of accesses to a page-table entry for a certain input, can result in lots of interrupt requests in accessed flags monitoring attack which can be detected by monitoring interrupt requests 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 [25, 26].

3.6.3 TLB flushing through Hyperthreading. 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 [25, 26].

3.7 DRAMA Attacks

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 [19, 26].

3.8 Cache-DRAM Attacks

Using two threads for launching the Prime+Probe attack [18, 24] and DRAMA attacks [19] 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 [26] (Fig. 6).

3.9 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,

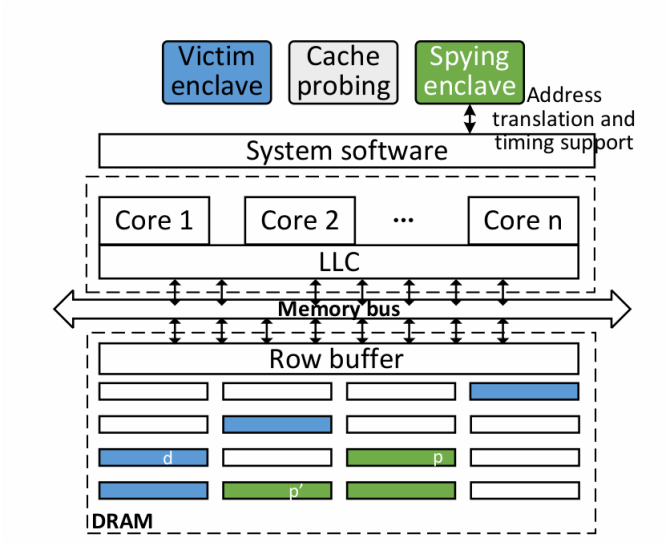

Fig. 6. Cache-DRAM attack [15].

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 [14].

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 Attack Detection

One of the methods that lead to the prevention and defense against the attacks is to detect the attack and stop executing completely, or while the attack is in action.

Chen et al. [3] 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. 7). 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. [23] do a statistical study on the percent of leakage, vulnerability, possibility and effectiveness of page-fault based attacks which they call pigeonhole attacks. Then, they

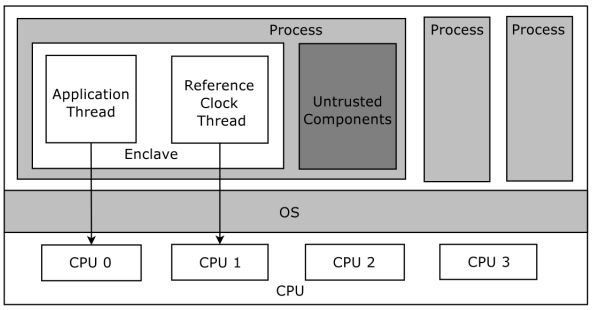

Fig. 7. System architecture of Déjà Vu. Blocks in gray are untrusted, which include the untrusted components of the processes and the entire OS kernel [3].

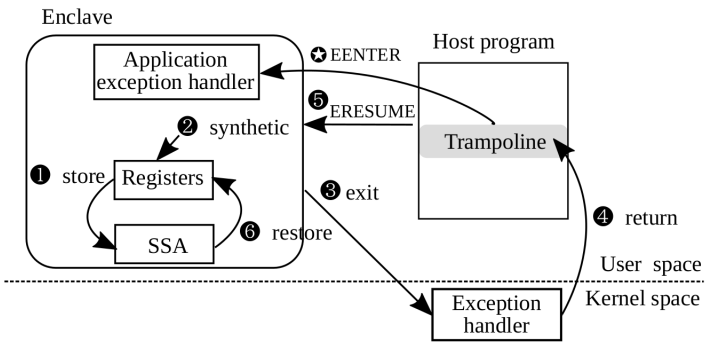

Fig. 8. Steps of an SGX asynchronous enclave exit (AEX) [22].

study 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 the physical 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. Based on the fact that the malicious OS cannot differentiate accesses which are occurred in a single page, they claim that by mapping all the input dependent data into a single page, the program will access the same pattern of the pages independent of the input and makes 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).

Shih et al. [22] make use of Intel transactional memory technology to isolate enclave code from interrupts caused by intentional page-fault based side-channel attacks performed by a privileged OS. As described in Fig. 8, the method includes 6 steps:

(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.

4.2 Attack Prevention

It is known that in a multicore system, 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 [13].

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 environment. So, there is a need to introduce a hack for indirectly doing this which has been implemented using Intel TSX by Gruss et al. [13].

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 [13].

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 [13].

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 [13].

ALGORITHM 2: Cloak sample code [13]

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

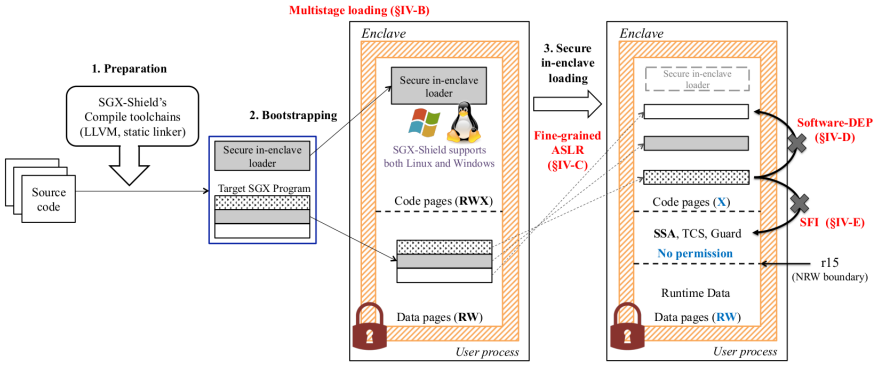

Fig. 9. Overall workflow of SGX-Shield: 1) the preparation phase builds a SGX binary from the target program's source code; 2) the bootstrapping phase loads the secure in-enclave loader into code pages and the target SGX program into data pages; and 3) the secure in-enclave loading phase finally loads the target SGX program [21].

Seo et al. [21] 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. [20] implement a block level memory controller as a new library combining ORAM [11, 12, 17] 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. [1] 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 [13] achieves noticeable protection against cache-based side-channel attacks. However, there will still be some leakage remaining which this research does not handle. Overall execution time can leak information. Also, 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 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 [22] 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.

The problem with Deja Vu [3] 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, attacks like B-SPM and HT-SPM (sneaky page monitoring attacks in general) which do not rely on AEX can still be effective [26].

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 [21] 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 [26]. 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. [23] 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 [26].

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, any solid 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.

6 CONCLUSIONS

In this report, we studied various kinds of attacks proposed by the past research and also reviewed the defenses in terms of technique, security, and performance and analyzed their effectiveness. In the end, we proposed new ideas for extensions of present defenses and the improvement of them. 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] 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).

[2] 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.

[3] 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.

[4] Intel® Corporation. [n. d.]. 4th Generation Intel® Core™ Processor. Retrieved April 12, 2018 from https://software.intel.com/en-us/node/524022

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

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

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

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

[9] Intel® Corporation. 2017. 6th Generation Intel® 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

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

[11] 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.

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

[13] 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*.

[14] 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.

[15] 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.

[16] Olga Ohrimenko, Michael T Goodrich, Roberto Tamassia, and Eli Upfal. 2014. The Melbourne shuffle: Improving oblivious storage in the cloud. In *International Colloquium on Automata, Languages, and Programming*. Springer, 556–567.

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

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

[19] 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.

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

[21] 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*.

[22] 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*.

[23] 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).

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

[25] 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.

[26] 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.

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