SGX-Bomb: Locking Down the Processor via Rowhammer Attack

Yeongjin Jang[*]
Oregon State University
yeongjin.jang@oregonstate.edu

Jaehyuk Lee
KAIST
jaehyuk.lee@kaist.ac.kr

Sangho Lee
Georgia Institute of Technology
sangho@gatech.edu

Taesoo Kim
Georgia Institute of Technology
taesoo@gatech.edu

Abstract

Intel Software Guard Extensions (SGX) provides a strongly isolated memory space, known as an *enclave*, for a user process, ensuring confidentiality and integrity against software and hardware attacks. Even the operating system and hypervisor cannot access the enclave because of the hardware-level isolation. Further, hardware attacks are neither able to disclose plaintext data from the enclave because its memory is always encrypted nor modify it because its integrity is always verified using an integrity tree. When the processor detects any integrity violation, it locks itself to prevent further damages; that is, a system reboot is necessary. The processor lock seems a reasonable solution against such a powerful hardware attacker; however, if a software attacker has a way to trigger integrity violation, the lock could result in a severe denial-of-service (DoS) attack.

In this paper, we introduce the SGX-Bomb attack that launches the Rowhammer attack against enclave memory to trigger the processor lockdown. The SGX-Bomb attack is simple yet alarming. Inside an enclave, this attack first finds conflicting row addresses at the same DRAM bank, and then repeatedly accesses them while bypassing the cache. If arbitrary bit flips have occurred inside the enclave because of the Rowhammer attack, any read attempts to the enclave memory results in a failure of integrity check so that the processor will be locked, and the system should be rebooted.

The SGX-Bomb attack is a serious threat especially to the public cloud providers who are supposed to run unknown enclave programs received from their clients, which might shut down their servers shared with other clients. We evaluate the effectiveness of the SGX-Bomb attack in a real environment with DDR4 DRAM; it takes 283 s to hang the entire system with the default DRAM refresh rate, 64 ms.

CCS Concepts

• **Security and privacy** → **Denial-of-service attacks**;

Keywords

Intel SGX; Rowhammer; DoS

[*]Work performed while the author was at the Georgia Institute of Technology.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SysTEX'17, October 28, 2017, Shanghai, China
© 2017 Copyright held by the owner/author(s). Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-5097-6/17/10…$15.00
https://doi.org/10.1145/3152701.3152709

ACM Reference Format:
Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking Down the Processor via Rowhammer Attack. In *SysTEX'17: 2nd Workshop on System Software for Trusted Execution , October 28, 2017, Shanghai, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3152701.3152709

1 Introduction

Trusted Execution Environment (TEE) is a promising approach to enable secure computation, which allows a secure execution of a program without relying on the underlying software stack such as an operating system and a hypervisor. Intel Software Guard Extensions (SGX) [18] is a commodity hardware-based TEE implementation designed to have a small trusted computing base (TCB); it only requires trustworthy components inside the processor package such as cache memory and memory controller (MC). Therefore, external hardware components such as the main memory (DRAM) and peripherals do not need to be trusted.

Relying solely on its small TCB, SGX provides isolated memory spaces called *enclaves* and aims to protect enclaves from two types of attacks. First, it thwarts software attacks on enclaves by preventing all the system software including operating system and hypervisors from accessing the enclave content. More specifically, any software-based unauthorized access to an enclave is prohibited by the page miss handler (PMH) and is redirected to the abort page [11]. All write attempts to the abort page are ignored, and all read attempts to it just return all-one value (i.e., `0xffffffffffffffff` for a 64-bit integer). Consequently, software attackers have no way to access enclaves.

Second, to prevent hardware-level data disclosure and forgery attacks (e.g., cold boot, memory bus snooping, or malicious RAM), Intel SGX adopts encryption and integrity checking schemes. The memory encryption engine (MEE) inside the MC ensures that all data transfers outside the processor package are encrypted (with AES-128); that is, all enclaves' data residing on the DRAM are encrypted. Thus, hardware attackers cannot disclose the plaintext data of enclaves without breaking the encryption scheme.

The MEE also maintains an integrity tree (a kind of a version tree) for the entire enclave memory space to cope with data forgery attacks. The integrity tree is updated whenever the MC transfers enclave data outside the processor and checked whenever the MC loads encrypted data residing on the DRAM into the processor. When the MEE detects any data corruption (i.e., integrity is broken), it will lock the MC and stop operating the entire processor to avoid further damages to enclaves, which is called as a *drop-and-lock* policy [15]. The MC and processor lock can be resolved only through a cold reboot, making the MEE generate new keys.

Because the threat model of SGX assumes that a violation of memory integrity can only be induced by hardware attacks only, the drop-and-lock policy seems a legitimate solution against hardware attackers who attempt to forge enclave content. However, we challenge to that assumption: *what will happen if a software-only attack undermine the integrity of an enclave by triggering a vulnerability in the memory hardware?* If the drop-and-lock policy is still applied to this case, the software-only attack can induce *halting the processor without having physical access to the DRAM.*

To answer this question, we construct the SGX-Bomb attack that launches the Rowhammer attack [21, 31], which can induce a physical disturbance error (i.e., bit flipping) in DRAM only by software, against the enclaves. To be specific, the SGX-Bomb attack identifies prerequisites of the Rowhammer attack in the enclave memory space and then induces random bit flipping in the memory by repeatedly accessing conflicting row addresses inside the same DRAM bank with cache flush and reload. By intentionally triggering the drop-and-lock policy to block the execution of the processor, the SGX-Bomb attack results in a system-wide denial-of-service (DoS); that is, the system is not responding any of interrupts; therefore the system should be *rebooted* to work properly.

We launched the SGX-Bomb attack to available hardware in the market; that is, an Intel Core i7-6700K processor equipped with Crucial 8GB DDR4 memory and installed as default settings. We observed that the entire system hang can be triggered in 283 seconds as the minimum.

The implication of the SGX-Bomb attack is alarming. From the viewpoint of a server (e.g., the public cloud), the expected use case of SGX is to securely run a program received from a remote client without knowing its details. This implies that a remote client can lock the server by launching the SGX-Bomb attack. Because system software cannot monitor the enclave by design and the lock is triggered at the hardware level, currently there is no defense against this attack. More importantly, the cloud infrastructure for SGX, in which its system availability is highly critical (i.e., directly impacting its profit), is a more severe target of the SGX-Bomb attack. It is because the processor lock will directly affect availability and service level agreements (SLAs), and its recovery takes long and consumes a lot of power, which is very costly in the cloud computing.

2 Background

In this section, we will describe the hardware modification in Intel Skylake processors for providing the integrity guarantee of SGX that is exploited by the SGX-Bomb attack. The confidentiality protection in SGX and the corresponding hardware modifications are not related to the SGX-Bomb attack. Thus they will not be illustrated in this paper. We direct the readers to [10, 11, 15, 19] for full description.

Processor reserved memory. Intel SGX introduces a new memory region called Processor Reserved Memory (PRM). The PRM is a physically continuous memory region located in the main memory (DRAM), and the BIOS determines its range at the boot time. The PRM region can be organized into two subregions: the Enclave Page Cache (EPC) and the integrity tree [8, 15]. All the data and code used by the enclave are stored in the EPC and the integrity tree maintains the MAC tags of the EPC data. All these operations are orchestrated by the memory encryption engine (MEE).

Memory encryption engine (MEE). The MEE is an extension of the processor's integrated memory controller (IMC), and the IMC routes all read and write requests to all EPC pages in PRM to the MEE [15]. Any access to EPC pages from non-enclave execution will be strictly prohibited by the MEE (against software attacks), and the MEE not only provides cryptographic protection over EPC pages for confidentiality but also verifies the integrity of the pages (against hardware attacks).

Integrity tree. The MEE provides data integrity and replay protection over EPC pages using the integrity tree [8, 15], which is an enhanced version of the classical Merkle tree [25] (*n*-ary Merkle tree with MAC using a hardware key). Tamper-resistance of the tree is guaranteed by storing the root node in processor-internal on-die SRAM. For every request, the MEE not only reads (writes) the content of the EPC but also verifies (updates) its MAC with (to) the integrity tree. Verifying MAC assures that the data read from EPC is not corrupted since the last time it was written from the processor to the DRAM.

Drop-and-lock policy. To prevent the forgery against EPC pages, the MEE enforces *drop-and-lock policy* [15, 18]. When the MEE detects any mismatch on the integrity tree, it immediately locks the memory controller to prevent further transactions from being serviced by the IMC. Eventually, this causes the *system to hang and reset must be required.*

3 The SGX-Bomb Attack

The objective of the SGX-Bomb attack is to lock the processor (i.e., launching a denial-of-service (DoS) attack) by intentionally triggering the defense mechanism of the MEE of SGX for physical attacks on memory integrity, through a software attack that utilizes the Rowhammer attack [21].

Because SGX is supposed to protect itself from any physical attack that tampers memory content in the enclave, the MEE follows the drop-and-lock policy [15], which locks the processor if the MEE observes any fault on the integrity check while loading and decrypting an encrypted memory block. Only an explicit system reboot can resolve the processor lock.

The drop-and-lock policy seems a valid solution against data forgery when only a hardware attacker can corrupt the EPC region. Unfortunately, we confirm that the Rowhammer attack can break this assumption through the experiment. When an enclave process launches the Rowhammer attack against its EPC pages, there is a chance to flip arbitrary memory bits in the EPC pages. That is, a software attack can corrupt data inside the EPC region.

The software-driven (or Rowhammer-driven) enclave data corruption also makes the processor halt because the MEE has no way to distinguish the data corruption caused by software from the data corruption caused by hardware. When an enclave process accesses flipped EPC pages, the MEE will detect integrity violation through discovering a mismatch of the MAC of the data with the integrity tree and then lock the MC (and the entire processor). It is because the memory modification through disturbance error is not reflected in the integrity tree. Once the lock is triggered the processor is neither able to execute instructions nor able to respond to any interrupt requests. The only way to regain the execution of the processor is to power down and up again.

This is a critical security problem because an unprivileged user-level program (i.e., in ring 3) in an enclave can lock the processor, leading to complete system down that makes the system software completely lose control of the system.

3.1 Threat Model

The SGX-Bomb attack is based on the following assumptions:

(1) The target machine is equipped with an Intel processor that supports SGX.

(2) The DRAM module of the target machine is vulnerable to the Rowhammer attack.
(3) The attacker has a user-level execution environment (i.e., ring 3) on the system and can execute a program in an enclave.
(4) The attacker has no system privilege (i.e., ring 0) of the target system and neither has a root-privilege account nor a kernel exploit to get the system privilege.
(5) The attacker does not have any physical access to the target machine.

We assume a realistic scenario of running a client application in SGX. For example, both running a program in SGX under the cloud environment and simply downloading a program for SGX over the Internet and executing it will fall into this model.

3.2 Attack Design

In this section, we describe the SGX-Bomb attack in detail. In particular, we would like to demonstrate the lock-down of the running processor by triggering a bit flipping on one of EPC pages, making the data integrity check fail. To launch this attack, we use the double-sided Rowhammer attack [31], which is known to be the fastest way to induce a bit flipping.

In the following, we describe in detail each step for launching the SGX-Bomb attack:

#1: Finding row addresses that reside in the same bank. The first step of the SGX-Bomb attack is to find conflicting rows that reside in the same bank in the EPC region because it is the requirement for launching double-sided Rowhammer attack. One way of finding such addresses is a method proposed in DRAMA [26], which reverse engineers the memory scramble function of the processor to figure out the bank mapping of a physical address. However, according to our threat model, we should not rely on this method because an attacker has no permission to get the physical addresses of the enclave memory space. Instead, we develop an exploit based on the timing information as a side channel for finding virtual addresses that map to the same bank in the DRAM within an enclave, as suggested in DRAMA [26] and the Google article [31].

Access of DRAM can be categorized in three ways in terms of timing: 1) access the same row, 2) access rows in different banks, and 3) access different rows (i.e., conflicting rows) in the same bank. Because the timing of accessing conflicting rows is slower than other two accesses, we can detect conflicting addresses only with virtual addresses by measuring and comparing the access time.

Figure 1 shows how we can find conflicting rows by exploiting access time as a side channel. We measure the timing of accessing two addresses `p1` and `p2` without hitting the cache for multiple times. Based on the timing categorization, we can easily observe the three clusters of timing if we scan a small number of heap space of an enclave that can accommodate the bank width of the memory (<256 KB for the tested DRAM). Then, we set the average value of the slowest and the median timing as the threshold (for accessing 1,000 times in i7-6700K, the threshold value was 600,000). After getting the threshold value, we can find conflicting row addresses by measuring the access time and comparing the timing with the threshold for a fixed `p1` and scanning `p2` for enclave heap space.

#2: Finding interleaved row addresses. The double-sided Rowhammer attack also requires an attacker to know two addresses in the same bank but interleaved by one row in the middle. Although we can find the row addresses in the same bank through step #1, we need to figure out two rows that are interleaved by one row. In other words, we need to have the addresses such that the first address points to *i*th row and the second address points to *i*+2th row.

```
// measure the timing of accessing two addresses p1 and p2
void enclave_access_row(uint64_t *p1, uint64_t *p2, uint64_t n_trial) {
  // run for n_trial times (to amplify the delay)
  while (n_trial-- > 0) {
    // flush two addresses from the cache
    asm volatile("clflushopt (%0)" :: "r"(p1) : "memory");
    asm volatile("clflushopt (%0)" :: "r"(p2) : "memory");
    asm volatile("mfence;");
    // access two addresses
    asm volatile("mov (%0), %%r10;" :: "r"(p1) : "memory");
    asm volatile("mov (%0), %%r11;" :: "r"(p2) : "memory");
    asm volatile("lfence;");
  }
}
#define N_THRESHOLD (600000)
#define N_TIMES     (1000)
// Runs outside of an enclave
bool check_addr_in_the_same_bank(uint64_t *p1, uint64_t *p2) {
  // returns ~500000 if p1 and p2 are in the same row
  // returns ~550000 if p1 and p2 are in different banks
  // returns > 600000 if p1 and p2 are in different rows
  // in the same bank
  size_t start_time = rdtscp();
  enclave_access_row(p1, p2, N_TIMES);
  size_t end_time = rdtscp();
  return( (end_time - start_time) > N_THRESHOLD );
}
```

Figure 1: A code snippet for finding two different row addresses that reside in the same bank. The function `enclave_access_row()` accesses two addresses `p1` and `p2` multiple times without hitting the cache (by `clflushopt`). The access time will be slower than the other accesses if `p1` and `p2` are located in different rows in the same bank. The function `check_addr_in_same_bank()` measures the access time of these addresses and detect conflicting addresses by a threshold value.

Because Intel SGX uses a fixed physical address range for the entire EPC region as well as the current Linux SGX driver has a simple virtual memory allocation algorithm, finding such interleaved row addresses are easy. By sequentially scanning the virtual address space of an enclave for finding conflicting rows, we can find such physically interleaved rows. In particular, when we set a sufficient (e.g., 16 MB) margin to the starting address for scanning the heap space, the scanned addresses are mapped into a sequential physical address space in the EPC region. In this setting, the conflicting rows that are found in sequence in the virtual address space are also sequentially placed in the physical address space. In consequence, if we observe three conflicting virtual addresses in sequence, let say them as `p1`, `p2`, and `p3`, then all these addresses are highly likely to be in sequence in the physical address space. So `p1` and `p3` are the bank-conflicting row addresses with interleaving only one row that `p2` points to. This is because the current Linux SGX driver sequentially allocates EPC pages when initializing an enclave. Using this method, we can find the target addresses of the double-sided Rowhammer attack.

#3: Triggering bit flipping. The last step of the SGX-Bomb attack is launching the Rowhammer attack to the enclave address space. After having two target row addresses, we apply the double-sided Rowhammer attack by running the code shown in Figure 2. In the code, the function `dbl_sided_rowhammer()` probes two interleaved rows `p1` and `p2` for multiple times. After one round of hammering is finished, the function `chk_flip()` will read the entire heap addresses in the enclave to check the occurrence of any bit-flipping by the Rowhammer attack. If the routine observes any bit flips, the MC and the entire processor will be locked.

To amplify the chance of hammering, we run this code with multiple threads; because the processor that we tested has four physical cores, we run the attack with four threads to maximize the chance of bit flipping.

```
uint64_t *ptr = ENCLAVE_HEAP_ADDRESS;
size_t mem_size = SIZE_OF_ENCLAVE_HEAP;

void chk_flip() {
  for(uint64_t i=0ul; i<mem_size/sizeof(uint64_t); ++i) {
    ptr[i]; // read memory
    // MC will be locked up if any bit in the read block is flipped.
  }
}
void dbl_sided_rowhammer(uint64_t *p1, uint64_t *p2, uint64_t n_reads) {
  while(n_reads-- > 0) {
    // read memory p1 and p2
    asm volatile("mov (%0), %%r10;" :: "r"(p1) : "memory");
    asm volatile("mov (%0), %%r11;" :: "r"(p2) : "memory");
    // flush p1 and p2 from the cache
    asm volatile("clflushopt (%0);" :: "r"(p1) : "memory");
    asm volatile("clflushopt (%0);" :: "r"(p2) : "memory");
  }
  chk_flip();
}
```

Figure 2: A code snippet for launching the double-sided Rowhammer attack in an enclave. The function `dbl_sided_rowhammer()` hammers two addresses `p1` and `p2` for multiple times. After one round of hammering is finished, the function `chk_flip()` will read the entire heap addresses in the enclave. The MC and entire processor will be locked if any bit flip is observed during the execution of this routine.

Refresh rate (ms)	**64** (default)	**128** (2×)	**256** (4×)	**503** (max)
Min. elapsed time (s)	283	30	4	1

Table 1: The minimum time taken to see the system hang by the SGX-Bomb attack by varying DRAM refresh rates. Because the chance of bit flipping by the Rowhammer attack increases as DRAM refresh time increases by, the minimum time for observing system lock is reduced for the longer refresh time.

4 Attack Evaluation

We implemented and launched a proof-of-concept of the SGX-Bomb attack on a target system equipped with an Intel Core i7-6700K processor and Crucial 8GB DDR4 memory. For the evaluation, we measured the minimum required time taken to trigger the system lock and analyzed the effect of the attack.

4.1 Attack Timing

Table 1 shows the minimum time taken to observe system hang while launching the SGX-Bomb attack for various DRAM refresh rates. The attack timing is not stable because the Rowhammer attack is caused by physical characteristics of a DRAM module, and the rate of bit flipping occurrence depending on the physical status. Thus, we show only the minimum timing that we observed for the successful attack to demonstrate its feasibility.

Normal DRAM refresh timing. For the normal DRAM refresh rate, which is 64 ms and 7.8 μs per each row, the earliest time taken for observing the system hang was 283 s.

Adjusted DRAM refresh timing. To simulate a highly vulnerable DRAM module, we adjusted the DRAM refresh rate. Reducing the DRAM refresh rate made the SGX-Bomb attack more effective. For doubling the refresh rate (128 ms), we observed the system lock in around 30 s. For the slower refresh rates, 256 ms (four times than normal) and 503 ms (the maximum rate on the processor), we observed the system lock in 4 s and 1 s, respectively.

4.2 Validating Processor Lock

We validated the locking of the entire processor by sending requests that the system supposed to give a response while launching the attack. Such requests include:
(1) Typing keystrokes on the console (in ring 3),
(2) Connecting to a network service such as `sshd` (in ring 3, with root privilege),
(3) Generating keyboard interrupts (`CTRL+ALT+DEL` or `sysrq` keys, in the kernel, ring 0),
(4) Pressing the reset button (physical).

From (1) to (3), for both ring 3 and ring 0 request handlers, we cannot observe any response from the system; that is, even for the system software including kernel cannot recover from the hang triggered by the SGX-Bomb attack. In the case of (4), we observed a different behavior than normal when the processor is locked. On our system, pressing the reset button on the system immediately reboots the system starting from the BIOS screen. This is done by the motherboard sending a signal to the reset pin of the processor, and the processor will initiate reboot upon receiving the signal. In contrast, when the processor lock is triggered, the system waits for around five seconds then the system completely turns its power off and turns it on again in order to initiate a cold reboot process. We believe that the power down is caused by the motherboard because the motherboard did not get any response for signaling on the reset pin of the processor, implying that the reset signal does not work at all while the processor is locked. Thus, the motherboard tries the physical reboot process that powers off and on the entire system. From this result, we think that the processor requires a complete power down and power up to be recovered by the lock.

5 Discussions

In this section, we explain why the SGX-Bomb attack is a serious security problem and describe potential software- and hardware-based countermeasures against SGX-Bomb.

Implication of SGX-Bomb for the cloud provider. SGX-Bomb is a serious threat to cloud providers because a system lock caused by the attack requires an explicit system reboot for their server shared by a number of customers. In particular, a system reboot is a serious problem to cloud providers because some of the popular cloud services, such as `Memcached` and `Redis`, require a warm-up phase longer than one hour [12], and, obviously, a faster `kexec`-based system reboot mechanism [20] cannot be applied when the MC (and the processor) is locked.

More importantly, availability is an essential requirement of the cloud service. For example, Amazon EC2 gives its customers service credits when its service does not meet the guaranteed 99.95% of monthly uptime [1]. Further, an unavailable cloud service would affect other services as shown by a recent outage of Amazon Web Services that resulted in taking down other services on the Internet as a chain reaction [33]. To avoid such problems, the cloud providers have to maintain many replicas, increasing their service costs.

Exploiting Rowhammer for SGX-Bomb is easier. The SGX-Bomb attack is straightforward to perform in comparison with other Rowhammer attacks because SGX-Bomb does not require a sophisticated preparation phase for a successful attack. The only requirement to halt the processor in the SGX-Bomb attack is to flip *any* memory bits inside the entire EPC region; that is, the attack surface is wide enough to contain a vulnerable memory cell (e.g., up to 128 MB for SGX version 1). In contrast, the other Rowhammer attacks are hard to launch due to their complex attack requirements. For example, an attack that aims to flip specific bits controlling privilege (e.g., making a page table entry writable by a user process [31]) not only requires controlling of memory allocation to place the target memory pages into vulnerable DRAM rows, but also requires flipping a specific bit (or bits) for a reliable attack [34].

SGX makes known Rowhammer detection schemes obsolete. The strong confidentiality guarantees provided by Intel SGX make the SGX-Bomb attack covert against static and dynamic analysis. First, the system software has no way to statically inspect the source code or binary that will be executed inside an enclave, when it dynamically loads its (encrypted) binary [29]. A two-way sandbox [17] that restricts instructions an enclave process can use (e.g., `clflushopt`) is a possible mitigation; however, the Rowhammer attack is still possible without relying on such instructions [14].

Second, system software cannot dynamically analyze and profile an enclave process because it is neither able to access the enclave memory nor able to retrieve performance events of the enclave process. The Rowhammer attack generates many cache misses because it has to directly access the memory to trigger a fault. Inspired by this characteristic, ANVIL [2] and a Linux kernel patch [9] profile cache misses using performance monitoring units (PMUs) to detect the Rowhammer attack. However, Intel SGX has a feature to do not accumulate performance monitoring events directly associated with an enclave for confidentiality, known as the anti side-channel inference (ASCI) [19]. Because the ASCI hides the number of cache misses generated by enclaves, both detection methods cannot detect an enclave conducting the Rowhammer attack.

Additionally, we confirm that Intel SGX can hide other Rowhammer attacks as well. We observe that the ASCI hides the cache misses of an enclave even when it accessed normal, non-EPC pages. This implies that SGX-based Rowhammer attacks against non-enclave memory (e.g., page table entries) are hidden from the system software, realizing *covert* Rowhammer attacks.

The root-cause is vulnerable DRAMs. The root-cause of the SGX-Bomb attack is DRAMs that are vulnerable to the Rowhammer attack and is not a design flaw of Intel SGX. Hence, the fundamental solution to block the SGX-Bomb attack is using Rowhammer-free DRAM or developing countermeasures to them. For instance, a DRAM module that supports target-row refresh (TRR) [28] could be an effective defense. In addition to TRR, Intel developed a method called pseudo-TRR (pTRR) [3] that works similar to TRR but does not require drastic changes in DRAM in their recent processors. However, both approaches require DRAMs to be compliant to them, and vulnerable DRAM modules are still available on the market; it is unclear whether future DRAM modules, which are becoming dense and stacked, are free from the Rowhammer attack.

Is error-correcting code (ECC) memory effective?. The ECC memory can mitigate the SGX-Bomb attack by mitigating the Rowhammer attack, but it cannot be a solution. This is because a previous work [22] confirmed that attackers could flip more than one bit in a memory block. Since the ECC memory can only correct 1-bit error and detect 2-bit error, the ECC memory cannot detect nor prevent attacks that flip more than 3-bits in a single memory block, which is totally possible by the Rowhammer attack. Even for detecting 2-bit error, without having a new policy different than drop-and-lock, the SGX-Bomb attack can still be possible.

Potential software-based mitigation. Since processors supporting Intel SGX (i.e., all Skylake and Kaby Lake processors) and DRAM modules vulnerable to the SGX-Bomb attack are already widely deployed on the market, we highly demand software-based mitigation against the SGX-Bomb attack to prevent any further damages.

To this end, we are considering two possible mitigation approaches. First, we plan to modify the Linux SGX driver to do not sequentially allocate EPC pages in the physical address space or to prevent allocations on weak cells in the DRAM. As explained in §3.2, the SGX-Bomb attack can easily detect conflicting DRAM rows because the current Linux SGX driver sequentially allocates EPC pages. Similar to the approach taken in Brasser et al. [5], modifying the driver to do shuffling/blocking during EPC page allocation phase make detecting interleaved conflicting rows difficult.

Second, we plan to use *uncore* PMUs (e.g., cache box and IMC PMUs) to detect whether the SGX-Bomb attack is being performed. We confirm that the ASCI does not hide performance events observable by uncore PMUs, as mentioned by Costan et al. [11]. We think that this is because differentiating and filtering enclave's performance events at the uncore PMUs is difficult to implement in hardware. We plan to investigate all the performance events of the SGX-Bomb attack that can be monitored by uncore PMUs to implement a detection method.

Better defense schemes than just processor lockdown. To avoid a serious DoS problem caused by possibly vulnerable DRAM, the processor may require different policies to take when it detects enclave data corruption, instead of the drop-and-lock policy. One possible policy is disabling all further SGX operations when there was any data corruption in an enclave. This policy also brings a DoS problem, but, at least, not the entire system halts and non-enclave processes work normally. Additionally, if the processor has a way to detect which enclave process is performing the SGX-Bomb attack (e.g., counting the number of cache misses each enclave generates) or maintains individual integrity trees for different enclaves, the processor can selectively remove the problematic enclave and let the system software know about it. However, implementing such features inside the processor would not only increase hardware costs but also experience false positives.

6 Related Work

This section, we introduce existing Rowhammer attacks and several countermeasures against them. We also introduce other attacks against Intel SGX.

The Rowhammer attack. Kim et al. [21] found that a DRAM disturbance error can be triggered by software repeatedly accessing (or hammering) DRAM rows in the same bank, which results in bit flipping in memory. The Google Project Zero team [31] turned this observation into a real exploit by inducing a bit flip into a page table entry (PTE), which allows an attacking process to manipulate its page tables freely. Because of its tremendous influence, may succeeding researches have been conducted. Gruss et al. [14] and Bosman et al. [4] have shown that a remote entity can launch the Rowhammer attack through JavaScript. Xiao et al. [36] and Razavi et al. [27] have shown that the Rowhammer attack is possible in cross-VM settings by attacking Xen paravirtual memory management or exploiting memory deduplication, respectively. Further, van der Veen et al. [34] have proposed a deterministic Rowhammer attack against an ARM platform. ANVIL [2] and a Linux kernel patch [9] use PMUs to check visible events of the Rowhammer attack such as high LLC miss rate. However, because of the ASCI feature of Intel SGX [19], such approaches cannot observe SGX-Bomb.

Attacks against Intel SGX. Researchers have considered possible attacks against Intel SGX to know whether it is secure enough and suggest improved security design for its future versions. Many side-channels attacks exploiting page fault [7, 32, 37], cache timing [6, 13, 16, 30], and branch prediction [24] have been proposed to attack SGX. Further, researchers have shown that it is possible to attack enclaves with software vulnerabilities [23, 35]. Note that existing attacks against Intel SGX focus on whether Intel SGX protects the confidentiality of an enclave from attackers. To the best of our knowledge, SGX-Bomb is the first software-based attack to compromise the integrity of an enclave.

7 Conclusion

Intel SGX is a promising technology to realize a secure computing environment with confidentiality and integrity. However, as shown in this paper, its strong integrity guarantee could become a double-edged sword if a software attacker intentionally triggers integrity violation through the Rowhammer attack in order to launch a severe DoS attack. Although we believe that using Rowhammer-free DRAM (if it exists) is the only complete solution against the SGX-Bomb attack, the software-based mitigation approaches that we are currently developing as well as the hardware design changes that we suggested would practically mitigate the problem.

Acknowledgement. We thank the anonymous reviewers for their helpful feedback. This research was supported, in part, by the NSF award DGE-1500084, CNS-1563848, CNS-1704701 and CRI-1629851, ONR under grant N000141512162, DARPA TC (No. DARPA FA8650-15-C-7556), and XD3 programs (No. DARPA HR0011-16-C-0059), ETRI IITP/KEIT[B0101-17-0644], NRF-2017R1A6A3A03002506, and gifts from Facebook, Mozilla and Intel.

Responsible vulnerability disclosure. To resolve the newly discovered security threat, we confidentially reported the SGX-Bomb attack to Intel by sharing a preliminary version of this manuscript in February 2017. The comment from Intel was that the cause of the SGX-Bomb attack is vulnerable DRAMs. We hope that this paper alarms the public regarding the risk and implication of the rowhammer attack against SGX.

References

[1] Amazon. 2017. Amazon EC2 Service Level Agreement. (2017). https://aws.amazon.com/ec2/sla/.

[2] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks. In *Proceedings of the 21st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Atlanta, GA.

[3] Kuljit Bains, John Halbert, Christopher Mozak, Theodore Schoenborn, and Zvika Greenfield. 2015. Row Hammer Refresh Command. (Aug. 25 2015). US Patent 9,117,544.

[4] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida. 2016. Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (Oakland)*. San Jose, CA.

[5] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. 2017. CAn't Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory. In *Proceedings of the 26th USENIX Security Symposium (Security)*. Vancouver, BC, Canada.

[6] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *Proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT)*. Vancouver, BC, Canada.

[7] Jo Van Bulck, Nico Weichbrodt, R. 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 (Security)*. Vancouver, BC, Canada.

[8] S. Chhabra, U.R. Savagaonkar, M.A. Goldsmith, S.P. Johnson, R.M. Leslie-Hurd, F.X. Mckeen, G. Neiger, R. MAKARAM, C.V. Rozas, A.L. Santoni, et al. 2016. Secure memory repartitioning. (Aug. 3 2016). https://google.com/patents/EP3049992A1?cl=no EP Patent App. EP20,140,849,831.

[9] Jonathan Corbet. 2016. Defending against Rowhammer in the kernel. (2016). https://lwn.net/Articles/704920/.

[10] Victor Costan, Ilia Lebedev, Srinivas Devadas, et al. 2017. Secure Processors Part I: Background, Taxonomy for Secure Enclaves and Intel SGX Architecture. *Foundations and Trends® in Electronic Design Automation* 11, 1-2 (2017), 1–248.

[11] Victor Costan, Ilia Lebedev, Srinivas Devadas, et al. 2017. Secure Processors Part II: Intel SGX security analysis and MIT Sanctum Architecture. *Foundations and Trends® in Electronic Design Automation* 11, 3 (2017), 249–361.

[12] Aakash Goel, Bhuwan Chopra, Ciprian Gerea, Dhrúv Mátáni, Josh Metzler, Fahim Ul Haq, and Janet Wiener. Fast database restarts at Facebook. In *SIGMOD14*.

[13] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache Attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security (EuroSec)*.

[14] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A Remote Software-induced Fault Attack in JavaScript. In *Proceedings of the 13th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[15] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processor. *IACR Cryptology ePrint Archive* 2016 (2016), 204.

[16] M. Hähnel, W. Cui, and M. Peinado. 2017. High-Resolution Side Channels for Untrusted Operating Systems. In *Proceedings of the 2017 USENIX Annual Technical Conference (ATC)*. Santa Clara, CA.

[17] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Savannah, GA.

[18] Intel. 2015. SGX Tutorial, ISCA 2015. http://sgxisca.weebly.com/. (June 2015).

[19] Intel. 2016. Intel 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C and 3D. (Sept. 2016).

[20] Sanidhya Kashyap, Changwoo Min, Byoungyoung Lee, Taesoo Kim, and Pavel Emelyanov. 2016. Instant OS Updates via Userspace Checkpoint-and-Restart. In *Proceedings of the 2016 USENIX Annual Technical Conference (ATC)*. Denver, CO.

[21] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. In *Proceedings of the 41st ACM/IEEE International Symposium on Computer Architecture (ISCA)*. Minneapolis, MN.

[22] M Lanteigne. 2016. How rowhammer could be used to exploit weaknesses in computer hardware. (2016). http://www.thirdio.com/rowhammer.pdf.

[23] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent B. Kang. 2017. Hacking in Darkness: Return-oriented Programming against Secure Enclaves. In *Proceedings of the 26th USENIX Security Symposium (Security)*. Vancouver, BC, Canada.

[24] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *Proceedings of the 26th USENIX Security Symposium (Security)*. Vancouver, BC, Canada.

[25] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 369–378.

[26] Peter Pessl, Daniel Gruss, Clementine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM addressing for cross-CPU attacks. In *Proceedings of the 25th USENIX Security Symposium (Security)*. Austin, TX.

[27] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. 2016. Flip Feng Shui: Hammering a needle in the software stack. In *Proceedings of the 25th USENIX Security Symposium (Security)*. Austin, TX.

[28] Perry Willmann Remaklus Jr and Robert Michael Walker. 2007. Method and system for providing independent bank refresh for volatile memories. (Feb. 27 2007). US Patent 7,184,350.

[29] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*. San Jose, CA.

[30] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Proceedings of the 14th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*.

[31] Mark Seaborn and Thomas Dullien. 2015. Exploiting the DRAM rowhammer bug to gain kernel privileges. In *Black Hat USA Briefings (Black Hat USA)*. Las Vegas, NV.

[32] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing Your Faults From Telling Your Secrets. In *Proceedings of the 11th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*. Xi'an, China.

[33] Chris Smith. 2017. Amazon finally explained what happened when it accidentally took down the internet. (2017). http://bgr.com/2017/03/02/why-internet-isnt-working-amazon-aws-outage/.

[34] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. 2016. Drammer: Deterministic Rowhammer attacks on mobile platforms. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security (CCS)*. Vienna, Austria.

[35] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In *Proceedings of the 21th European Symposium on Research in Computer Security (ESORICS)*. Crete, Greece.

[36] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and MR Teodorescu. 2016. One bit flips, one cloud flops: Cross-VM row hammer attacks and privilege escalation. In *Proceedings of the 25th USENIX Security Symposium (Security)*. Austin, TX.

[37] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (Oakland)*. San Jose, CA.