Foreshadow attack

Alexandru Mocanu
Computer Science Department, University of Bucharest, Romania

Abstract

Memory isolation represents a key component in how computers are built and became even more important with the increasing usage of the cloud technologies, where multiple distrusting entities can share the resources of a machine. In this paper, we make an overview of an example of hardware enforced trusted execution environment (TEE) that offers both confidentiality and integrity for the code that is executed and the used data, namely Intel's Software Guard eXtensions (SGX). We also present the Foreshadow attack that manages to breach the security requirements of the SGX, that can be used even by unprivileged users.

Then, we touch upon the evolution of this initial attack called Foreshadow Next Generation (Foreshadow NG), which was discovered by Intel during investigations regarding the Foreshadow attack. The second attack shows that the same structure from the first attack can be used in a different context to achieve the same goal, which is to breach the memory isolation.

1 Introduction

The memory separation in a system showed serious flaws that go deep into the architecture of the processors with the appearance of attacks such as Meltdown[3] and Spectre[2] that took advantage of transient out-of-order instructions to read unauthorized memory. In the light of this events, Intel claimed that their SGX technology is not affected by the former attack and the latter can be avoided if coding guidelines are followed by developers which aimed to remove code gadgets and side-channel vulnerabilities.

However, more research on the Intel technology showed the possibility of a Meltdown inspired attack called Foreshadow[4], which moreover didn't use any assumptions on the code of the victim. Further investigations revealed that the root cause which allowed this attack to take place is the handling from the L1 cache system. This in term led Intel to refer to these attacks as L1 Terminal Fault (L1TF)[1], but they are also referred to as Foreshadow NG[5] by the authors of the initial attack.

In the next section, we present how the SGX enclave mechanism works with emphasis on how the confidentiality is preserved and how the integrity is validated. Then, we resume the Foreshadow attack on the SGX enclave and understand how it affects the SGX enclaves as well as its general form, Foreshadow NG. In section 3, we discuss the preventive measures against these attacks. In section 4, we illustrate experimental results from papers on Foreshadow that reveal its effectiveness and also that it is crucial to bring the secret data into the L1 cache in order to mount the attack. Lastly, we offer some conclusions regarding the Foreshadow attack and its mitigation.

2 Technical description of the problem

2.1 SGX

SGX is a hardware based execution environment that is designed to work on a machine on which not even the operating system can be trusted. The main idea is to encrypt data and code of the desired application and realize the decryption only inside the processor through a memory encryption engine (MEE). Thus, when accessing the main memory the page will first be deciphered before being used (and also the data will be stored as plaintext in the cache), only if the process passes an authentication. The authentication mechanism is based on the concept of enclave, the general idea being

that only accesses from within the enclave, from the application's code, can actually read the data. Assuming that a process tries to read data from outside the enclave, the CPU will return value -1 regardless of the value found there, action called *abort page semantics*. Also relevant for this attack is that this process happens only after walking through the page table to ensure that the process has mapped the respective page.

Regarding the integrity of the code, SGX also offers the possibility to generate an attestation which validates that the executed code is the intended one. When creating a new enclave, two values are computed and stored at a secure location in the processor, inaccessible to any process: hash values MRENCLAVE (obtained from the initial code) and MRSIGNER (used to identify the enclave's author). Should someone want to obtain a certification, they can launch a challenge to the enclave. This will result in generating a report based on its code and used data that afterwards will also be signed using an attestation key (meant to guarantee the identity of the machine). This report will be sent back to the requester who can validate its authenticity using an Intel Service acting as a trusted third party in the process.

One more important aspect of the SGX architecture is that not only the application code is encapsulated in enclaves. Some of the components of the authenticity mechanism are also built as enclaves, namely the launch enclave (which creates and boots the other enclaves), the provisioning enclave (which manages the long-term attestation keys) and the quoting enclave (which is responsible for signing attestation reports). A key aspect regarding this architecture is that being able to break an enclave will also make this implicitly created enclaves vulnerable, which is also exactly the case of the Foreshadow attack.

2.2 Foreshadow attack

The basic attack is formed of three phases, the last two being represented by a classic Meltdown attack. In the second phase, an attacker tries to read the enclave data, thus generating an exception which leads to a race condition between the exception's detection and the attack formed by the transient instructions that attempts to send the secret via a side channel. Similarly to the Meltdown attack, a timing method is used to retrieve the secret. Precisely, an array will be used and in the transient instructions an access to the element from the array having equal index to the value of the secret will be made. In the third phase, the attacker can time the access for each element of the array, keeping in mind that the element corresponding to the secret should be cached due to the previous access and therefore its access time will be significantly smaller.

Only applying phases two and three will not result in any harmful behaviour since reading the memory of an enclave from outside will not result in a trap, therefore there will be no race condition. To be able to use the transient out-of-order instructions from the Meltdown attack, we need to create a trap when we are trying to read the secret data. For this, the authors of the Foreshadow attack propose a preparation phase one. The innovative idea takes advantage of the fact that the abort page semantics verification is made only after searching the required page in the page table and clears in the page table the bit "present". This action will not be detected as a malicious one by the operating system since if a page is not marked as "present", the operating system is free to use the rest of the memory as necessary. Also, similar to the behaviour of the Meltdown attack, clearing the "present" bit will now result in a page fault, triggering an exception. Another requirement to make the attack work is to bring the secret data to the L1 cache. A straightforward solution is executing the victim enclave in the first step, but other two variations of the attack are proposed. First is a framework allowing an attacker to execute step by step the enclave code and to read its memory after each instruction by cleverly manipulating the access of the enclave to the code zone. This variation can also be used by a non root user, which is to be more concerning . The second solution is to execute a concurrent attack in which a *spy thread* runs on a sibling logic core with the victim such that they will share the L1 cache.

2.3 Foreshadow NG

After further research on the initial attack, it was discovered that the general structure of it can also be applied in other contexts. Essentially, through manipulation of the memory mapping at different

levels, Foreshadow can be applied to breach the confidentiality of the OS, of a virtual machine (VM) or a hypervisor and of the SGX. For a better understanding on how this attack manifests at different levels, let us have a look at how the memory mapping works and what can go wrong.

In the context of the OS, a mapping is created for each process, called a page table, from its virtual memory pages to their physical location. Under a normal run, when a process requires one of its pages, the OS will look into the page table and bring it from the main memory. To avoid a high latency, frequently used pages are stored in cache and whenever they are requested again, another access to the main memory will not be needed. To boost this process even more, Intel is doing a parallel check on the page table and in the L1 cache. If a tag match is found between the requested entry from the page table and a data from the L1 cache, then the data will be sent immediately to the CPU, even if the checks on the page table are not finished. Even if the architectural effects will be reverted in the case of a page fault (which can potentially be caused by a variety of flags), this behaviour opens an window of opportunity for a Foreshadow type attack using transient instructions. Therefore, if in the page table a page is marked as not *present*, similarly to the initial attack, but the other contents of the page table entry were not cleared, then an attacker can read the data from that page if it is also located in the L1 cache.

Going one level deeper in the architecture, we take a look at how mapping of the virtual memory works in the case of a process from a VM. Since inside a VM, we also have an operating system we will also see the page table presented previously, but instead of pointing virtual memory of the process to locations from main memory, a page table from a VM points to locations from its own virtual memory. And afterwards an Extended Page Table (EPT) − that is specific to the VM and not to each process in it − is used to map the virtual memory of the guest to the main memory of the host. The page table walk from the VM works similarly to the one presented in the context of the OS, meaning that data from the L1 cache will be read even before checking the EPT. With that in mind, an attacker can manipulate the page table of its process and access pages from outside the guest virtual machine that resides in the L1 cache. Because the checks are delayed, information can be extracted through transient instructions. It is worth mentioning that also the third level in the architecture, namely the abort page semantics mechanism in the context of the SGX can be used inside a VM, meaning that an attacker can break the security of the enclaves even from a guest VM.

3 Possible solution

Regarding the mitigation of the initial Foreshadow attack, a possible solution is to offer hardware-level patches that can treat more carefully the transient instructions especially in the light of the appearance of the other similar type of attacks, Spectre and Meltdown. Apart from that, software solutions should also be found in order to be able to protect systems that are already in use. The solutions offered by the authors of the attack were to avoid using HyperThreading on two logical cores that execute code from within the enclave, respectively from outside of it and to flush the L1 cache upon exit from the enclave. The former aims to mitigate the concurrent attacks, whilst the second is mitigating the attacks happening after the enclave finished executing. The initial attack also allowed a root attacker to use a set of instructions that allow flushing and reloading data into the L1 cache, for reading the memory of an enclave without even having executed it. Detecting malicious behaviour when using those commands also represents an important mitigation of the Foreshadow attack, but not crucial since that type of attack can only be mounted by root adversaries.

Looking at the Foreshadow NG, we encounter two somewhat different attacks regarding the capabilities of an attacker. When mounting an attack from a virtual machine, the attacker can control the pages that he's accessing as long as they reside into the L1 cache. Therefore, the same ideas of mitigation as for SGX can be applied, meaning that we want to avoid sharing the L1 cache with malicious parties. To achieve this goal, flush instructions should be used before a guest VM or the hypervisor finish their execution to eliminate possible secrets from the cache. Also, in the interest of eliminating the possibility of a concurrent Foreshadow attack, disabling the HyperThreading or running the hypervisor thread on different cores than a po-

tential dangerous guest VM is recommended.

In the context of a Foreshadow attack applied directly to the OS, an attacker cannot control the page table from a user application being therefore restricted to using only page table entries that already exist, but are not normally accessible, for instance a page that was swapped out of the main memory. In this example, the page should not be available. Though, if the page table entry has not been entirely cleaned, meaning that only the *present* bit is unset, then an attacker can take advantage and read that page in the event that it is still located in the L1 cache. The idea to take away from here is that setting bits is not enough in this context and a complete reset approach should rather be adopted. The solution proposed by the authors of the Foreshadow attack is to use the fact that page tables for most systems can reference more addresses than the actual number of the physical addresses and thus we can always set the value in the page table entry to an invalid physical address by setting some of the first bytes from the address.

4 Experiments and Results

To illustrate the effectiveness of the initial Foreshadow attack, the authors made an evaluation of the success rate for an unprivileged user and for a root user. In both scenarios, the attack is made with and without Transactional Synchronization eXtensions (TSX), which is an optimization for the attack that helps at handling exceptions without signaling them to the OS, in order to avoid such unnecessary context switches between kernel and user mode. So, in total, four scenarios were used. To test each of them, 200 runs of the process have been made, in each of them being selected 5 enclave pages from its total of 1024 as the ones holding the secret. The median success rate for extracting a cache line was as follows: 99.92% for root user with TSX, 97.32% for root user without TSX, 96.82% for unprivileged user with TSX and 81.14% for unprivileged user without TSX.[4] As expected, there is a higher success rate for root user since multiple optimizations can be applied and also the success rate is better when using TSX since making less context switches offers a better opportunity to extract the secret before it is being switched out from the cache. Another relevant observation is the evolution of the success rate from within a cache line. As expected, for all four scenarios, the success rate drops when the index of the byte from the cache line increases. That is because the more time it takes to extract the secret, the greater the chance of the secret being swapped out is.

Another interesting experiment presented in the Foreshadow paper[4] tried to understand how the attack behaves in a context in which the secret resides in the L2 cache, but not in L1 and in L3, but not in L1 and L2. The result one may expect would be to see the success rates drop since having the secret somewhere upper in the hierarchy will increase its access time offering a smaller window to extract it. Though, interestingly, the results for both experiments dropped from above 80% as in the L1 cache case to 0% for both L2 and L3. This steep decrease of the success rates have led the authors to believe that the cause is not only a timing difference between L1 and L2. It was later researched and confirmed by Intel that the issue came from a difference of microcode between the two of them.

5 Conclusion

To conclude, we see that Foreshadow attack is a highly effective attack which idea was based on the Meltdown attack and its concept of transient out-of-order instructions. The initial attack was designed to target the security of Intel's hardware enforced environment SGX and revealed serious flaws that affected both its confidentiality and integrity.

After an insightful research from Intel, it was found that the root cause that allow the Foreshadow attack is not related to SGX, but to how fetching the data from the L1 cache is done. Also, it is important that a more general context was found in which the Foreshadow NG attack can be applied.

An important idea to take away from this attack is that even if software solutions are proposed to mitigate this attack, the increasing usage of transient instruction in attacks such as Foreshadow, Meltdown and Spectre showed that something more fundamental should be changed in the architecture of CPUs to prevent further vulnerabilities.

References

[1] Intel. Intel Analysis of L1 Terminal Fault. https://software.intel.com/content/www/us/en/develop/articles/software-security-guidance/technical-documentation/intel-analysis-l1-terminal-fault.html, August 2018. Accessed: 2021-06-26.

[2] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[3] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[4] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium*. USENIX Association, August 2018. See also technical report Foreshadow-NG [5].

[5] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx, Thomas F. Wenisch, and Yuval Yarom. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. *Technical report*, 2018. See also USENIX Security paper Foreshadow [4].

‘