Software Side-Channel Attacks Against Intel SGX Enclaves

CS 658 Research Paper

Colin Howes
University of Waterloo
200 University Ave. W
Waterloo, ON N2L 3G1
chowes@uwaterloo.ca

ABSTRACT

Intel Software Guard Extensions (SGX) is a hardware-backed security extension of the Intel architecture that allows user-level software to run securely in an environment where all other software on the host system is untrusted. SGX uses secure enclaves running in Processor Reserved Memory combined with a software attestation scheme to provide confidentiality and integrity guarantees to users wishing to execute software on an untrusted remote system. SGX is vulnerable to a number of software side-channel attacks, which leverage performance measurements to determine memory access patterns and derive secrets from software executing in secure SGX enclaves. SGX vulnerabilities to software side-channel attacks, possible countermeasures, and directions for future research are discussed.

1 BACKGROUND

1.1 Intel SGX

Intel Software Guard Extensions (SGX) is an extension to the Intel architecture designed to allow programs to run securely in an environment where all software on the host machine is potentially untrusted [3]. SGX was designed to address the problem of *secure remote computation*, that is, secure execution of software on a remote system controlled by an untrusted party [3]. Under this threat model, *all* software on a remote system is potentially malicious, including privileged software such as the operating system and hypervisor. Thus SGX must provide protection to user-level programs from untrusted software running at user level as well as at higher privilege levels [4].

SGX provides confidentiality and integrity guarantees through the use of trusted hardware that sequesters a user-level program into a secure container called an *enclave*, and then proves to a user through *software attestation* that he or she is running unmodified software protected in an enclave by secure hardware [3]. In order to provide software attestation, the host machine provides a cryptographic signature certifying an enclave's measurement hash, which is computed on a measurement of its contents after it is loaded and initialized [4]. The remote user can then verify this signature against an endorsement certificate provided by the hardware manufacturer, and can refuse to load his or her data into an enclave whose measurement hash does not match an expected value [3, 4]. Under this threat model, the remote user need only trust the hardware manufacturer responsible for providing the secure hardware and endorsement certificate.

1.1.1 SGX Enclaves. The SGX security model is centered around maintaining security-sensitive information in secure containers isolated from the rest of the untrusted host system. SGX accomplishes this through the use of a contiguous range of protected memory called Processor Reserved Memory, which is accessible only via a set of specialized microcode instructions called from within an enclave [4]. Enclave contents and meta-data are stored in a subset of Processor Reserved Memory called the Enclave Page Cache, which is split into 4 KB pages. Management of enclave pages is delegated to the untrusted host OS or hypervisor, which can manage pages via a limited interface using SGX microcode instructions, but may not access this memory directly. Rather, enclave pages are encrypted while in DRAM, and are decrypted in hardware as they are loaded into the cache by the CPU. Furthermore, memory management decisions made by system software are tracked in order to maintain isolation of protected memory by verifying that enclave pages can only be accessed by enclave code executing in the enclave associated with a given enclave page [3, 4, 7].

1.2 Software Side-Channel Attacks

Side-channel attacks are a class of attacks that leverage information about the physical properties of a system in order to carry out an attack. While physical side-channel attacks against modern hardware are difficult and costly, requiring advanced tools and physical access to the victim machine, software side-channel attacks are inexpensive to deploy and can be executed by anyone with remote or local access to a system [4]. The software side-channel attacks discussed here exploit hardware and software implementation details to acquire information about memory access patterns, which can be used to infer secrets from an otherwise secure system [5, 8–10]. Software side-channel attacks are not included in the SGX threat model despite the potential danger these attacks pose in a cloud computing environment, and SGX enclaves are not currently protected against this class of attacks [2, 7].

2 ATTACKS AGAINST SGX ENCLAVES

A number of distinct software side-channel attacks have been demonstrated against SGX enclaves in proof-of-concept attacks. While enclave data is protected in Processor Reserved Memory and is isolated from both OS and user-level processes, memory management and scheduling is deferred to the untrusted OS and/or hypervisor, giving a malicious OS a great degree of control over a victim SGX process and providing fine grained measurement capabilities. Thus, a malicious OS is free to dictate precisely how and when a victim enclave program is scheduled, and on which logical cores. This high magnitude of control over the victim process can be leveraged to create powerful privileged software side-channel attacks against SGX enclaves.

Moreover, the protection and isolation guarantees provided by SGX extend to malicious programs, meaning that an adversary can use an SGX enclave to launch attacks against a victim enclave running on the same host system. Since even privileged services are oblivious to the contents and behavior of SGX enclaves, the host OS has no means to monitor its SGX enclaves for malicious behavior, and thus cannot protect honest user-level processes from malicious enclave behavior.

2.1 Cache Timing Attacks

Caches are small regions of high-speed memory used to store recently accessed data in order to reduce future access times. In modern computers with multiple cache levels, each core typically has its own first and second (L1 and L2) caches, with a third level (L3) cache, or lowest level cache (LLC) shared across all cores. When the CPU needs to access data, accesses may be resolved in a cache (a cache "hit"), or data may need to be fetched from main memory (a cache "miss"), at which point it is stored in the cache, evicting older data. Cache accesses are considerably faster than memory accesses that must be resolved in main memory, and so caching data can significantly reduce computation time by reducing reliance on relatively slow main memory accesses. Cache timing attacks exploit the timing differences between cache hits and misses to infer secrets kept by a concurrently executing program by analyzing the victim processes memory access patterns [4, 7].

At a high level, an attacker can take advantage of the difference in access speed by filling a shared cache with data, thereby evicting data used by a victim process, and measuring the time it takes to access this data after a context switch. The attacker can determine which regions of memory were accessed by the victim based on the time it takes to access the data it had previously placed in the cache. Since a block of memory is mapped to a region of cache memory based on its virtual memory address, the attacker can us a cache attack to infer information about the victim's memory usage patterns, which can be used to extract secrets such as encryption keys [5, 7].

The Prime+Probe attack is cache attack that can be used to attack a victim process by targeting a shared cache. In a *priming* step, the attack fills the entire shared cache, thereby causing the victim's cached data to be evicted. Next, the attacker waits for the victim process to run, at which point it makes memory accesses that result in attacker memory blocks being evicted from the shared cache. Finally, in a *probe* step, the attacker attempts to access its data, timing memory accesses to determine which blocks of memory were used by the victim process [5, 7].

Proof of concept attacks against a number of AES implementations have demonstrated that an attacker can launch powerful Prime+Probe cache attacks against SGX enclaves by taking advantage of root-level scheduling privileges and performance monitoring tools. By leveraging control over a malicious host OS to schedule a Prime+Probe attack program on the same core as the victim SGX enclave with all other software running on other cores, an attacker can eliminate noise caused by the execution of other processes changing the cache set. In addition, since the attacker can also control scheduling, the attacker process can frequently interrupt the victim, providing very good temporal resolution - that is, by reducing the time that the victim is allowed to execute before the attacker can probe for cache misses, the attacker can make stronger inferences about the timing and order of memory accesses by the victim process. Thus, by isolating the attacker and victim programs onto a private core with a shared L1 cache, an attacker with root level privileges can mount a cache attack with high temporal resolution by enforcing frequent interrupts of the victim process by the Prime+Probe attacker process [5, 7].

Moreover, a remote attacker can take advantage of the protections offered by SGX enclaves to conceal malicious code and carry out cache attacks against co-located victim enclaves using only user-level code and performance monitoring tools. Therefore, since the OS has no straightforward means of monitoring the behavior of enclaves hosted on the system, side channel attacks carried out from malicious enclaves can be concealed from antivirus software by the protections offered by SGX. Such an attack has been shown to be capable of extracting private keys from an enclave running a current RSA implementation [8].

2.2 Page Fault Attacks

Similar to cache timing attacks, controlled-channel page fault attacks allow an attacker to derive secrets from a victim program based on memory access patterns. Page fault attacks rely on an attacker having control over memory management, as is the case when an attacker controls a malicious OS on which a victim is executing code in an SGX enclave [2]. In order to carry out an attack, the attacker must first analyze the victim's source code or a binary in order to identify input-dependent data accesses and input-dependent control transfers. When the application is run, the attacker can restrict access to a set of the victim's pages, causing a page fault to be triggered whenever the victim attempts to access an address on a target page. Since the CPU must report at least the base address of the faulting page to the OS, the attacker can record sequences of page accesses and can

use this information to run an offline analysis and make inferences about the victim's secret data at page level (4 KB) granularity [9, 10].

Controlled-channel page fault attacks are feasible against SGX enclaves, since a privileged attacker has control over memory management, and SGX aims to support legacy appliations without requiring that they be modified to conceal private data from an attacker that can observe memory access patterns [10]. Moreover, a malicious OS can exert fine-grained control over page allocation to carry out a "pidgeonhole" attack against an SGX enclave. Here, the OS allocates only 3 pages to the victim enclave - a code page, a source page, and a destination page - which make up the "pidgeonhole set". Any memory accesses that fall outside of these pages will cause a page fault, revealing to the OS which data or code the application is attempting to access. The OS can therefore maximize the information leaked to the attacker via the page fault channel. This technique has been demonstrated against SGX enclaves running implementations of OpenSSL and Libgcrypt [9].

2.3 Branch Prediction Analysis

Modern processors use instruction *pipelining* to fetch and decode instructions while previous instructions are executing. In order to maintain efficiency when a branch is encountered, the processor predicts which instruction will be executed after the branch based on past behavior. The predicted next instruction is fetched before the processor has determined which branch must be taken, and if the prediction is correct pipelined execution can continue as normal. On a branch misprediction the processor must clear the predicted instruction and load the correct instruction before execution can continue. In either case, the outcome is used as the basis for prediction the next time the branch is encountered [1, 6].

An attacker can leverage this behavior to infer information about a victim process through a branch shadowing attack. Briefly, an attacker uses knowledge of a program's structure to construct a "shadow" program, which has branches at the same points in execution as the victim. The attacker can then interrupt the victim process after a branch is encountered and run the shadow program starting at the address of the branch most recently encountered in the victim program. The attacker can then use timing differences or performance monitoring tools to determine whether the branch predictions made by the processor based on the behavior of the victim led to correct or incorrect predictions of the shadow program's behavior, and can infer information about the victim's execution patterns [6].

SGX does not clear an enclave's branch history when performing an enclave exit, and a privileged attacker can take advantage of this by forcing frequent interrupts such that all branches taken by the victim can be determined by the attacker. In order to reduce the noise inherent in attempting to time differences in performance between correct and incorrect branch predictions in the shadow program, an attacker can make use of the Last Branch Record (LBR), a CPU

debugging feature that records the last branch taken and whether the CPU's prediction was correct. This feature is only available in debugging mode, so the attacker must set the shadow program to run in debugging mode and check whether the predictions based on the behavior of the victim led to correct choices in the shadow program. The attacker can therefore use this privileged debugging mechanism to determine precisely which branches were taken by the victim and infer information about the input to the victim program. Proof of concept attacks have demonstrated that this technique can reveal information about input and program behavior protected in SGX enclaves that cannot be detected by the controlled channel page fault attacks described above [6, 9].

3 COUNTERMEASURES

A number of countermeasures have been proposed, both for future development of secure hardware, and additional security measures that can be employed by security conscious developers working with the current implementation of SGX.

3.1 Compiler Techniques

3.2 T-SGX

3.3 SGX-Shield

3.4 ORAM

3.5 Sanctum

4 FUTURE DIRECTIONS

5 CONCLUSIONS

REFERENCES

[1] Onur Aciimez, etin Kaya Ko, and Jean-Pierre Seifert. 2007. On the Power of Simple Branch Prediction Analysis. In *Proceedings of the 2Nd ACM Symposium on Information, Computer and Communications Security (ASIACCS '07)*. ACM, New York, NY, USA, 312–320. https://doi.org/10.1145/1229285.1266999

[2] Intel Corporation. 2015. Tutorial Slides for Intel SGX. (2015). https://software.intel.com/sites/default/files/332680-002.pdf

[3] Intel Corporation. 2016. Intel Software Guard Extensions. (2016). https://software.intel.com/en-us/sgx

[4] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86. https://pdfs.semanticscholar.org/2d7f/3f4ca3fbb15ae04533456e5031e0d0dc845a.pdf

[5] Johannes Gtzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Mller. 2017. Cache Attacks on Intel SGX. ACM Press, 1–6. https://doi.org/10.1145/3065913.3065915

[6] 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. *n Proceedings of the 26th USENIX Security Symposium* (2017).

[7] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX Amplifies The Power of Cache Attacks. *arXiv:1703.06986 [cs]* (March 2017). http://arxiv.org/abs/1703.06986 arXiv: 1703.06986.

[8] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clmentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. *arXiv:1702.08719 [cs]* (Feb. 2017). http://arxiv.org/abs/1702.08719 arXiv: 1702.08719.

[9] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2015. Preventing Your Faults From Telling Your Secrets: Defenses Against Pigeonhole Attacks. *arXiv:1506.04832 [cs]* (June 2015). http://arxiv.org/abs/1506.04832 arXiv: 1506.04832.

[10] Y. Xu, W. Cui, and M. Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy*. 640–656. https://doi.org/10.1109/SP.2015.45