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 security sensitive 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 [2]. 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 [2]. 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 [3].

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 [2]. 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 [3]. 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 [2, 3]. 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 [3]. 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 allocate or free enclave pages via a limited predefined 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 [2, 3, 5].

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 [3]. 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 [4, 6–8]. 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 [1, 5].

1.2.1 Cache Timing Attacks. Caches are small regions of high-speed memory used to store frequently accessed data in order to reduce future access times. In modern computers with multiple cache levels, each core has its own first and second (L1 and L2) caches, with a third level (L3) cache shared across all cores. When the CPU needs to access data,

accesses may be resolved in the 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 [3, 5].

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 [4, 5].

The Prime+Probe attack is cache attack that can be used to attack a victim process by targetting 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 [4, 5].

1.2.2 Page Fault Attacks. Page fault attacks can be employed by a malicious operating system to determine when a program is accessing specific pages. Page fault attacks are similar to cache timing attacks, in that an attacker moves a large number of pages into memory and leverages kernel-level information about page faults to determine which pages are used by a victim process.

1.2.3 Branch Shadowing.

2 ATTACKS AGAINST SGX ENCLAVES

Both cache timing attacks and page fault 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. Thus, a malicious OS is free to dictate how and when a victim enclave program is scheduled, and on which logical cores. This high degree of control over the victim process can be leveraged to create powerful privileged software side-channel attacks against SGX enclaves.

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 as such cannot protect honest user-level processes from malicious enclave behavior.

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 T-SGX

3.2 Compiler Techniques

3.3 Sanctum

4 FUTURE DIRECTIONS

5 CONCLUSIONS

REFERENCES

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

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

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

[4] 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

[5] 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.

[6] 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.

[7] 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.

[8] 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