Cache Based Side Channels: Attacks and Defenses

Sathya Chandran Sundaramurthy
sathyachandr@mail.usf.edu

Abstract

The efficiency of a processor relies heavily on the use of cache memory. However, the sharing of same cache space between different processes opens up a wide possibility of attacks. Research over the years has exposed the use of side-channels as a new attack mechanism to extract sensitive information from a targeted victim process. The attack could be as severe as extraction of the private key of an encryption algorithm. My term paper will explore various types of side-channel attacks possible on cache memory and the evolution of defense techniques to mitigate the same. This proposal provides a brief overview of the problem, and a outline for the final submission. Being a graduate student in Computer Science foucsing on security this problem seemed to be an interesting one to write my term paper on.

I. Introduction

A cache is a block of fast RAM that sits inside a CPU die. Whenever a program refers to a memory address the processor first checks the cache to see if the contents of that memory address have already been loaded. This case is called a *cache hit*. If the contents are not available in the cache yet, called a *cache miss*, the contents are then loaded from memory onto the cache. Subsequent accesses to the same memory address will now be faster. There are three categories of cache memory classified based on their proximity to the CPU –*L1, L2, and L3* – with L1 being closest. Closer the cache to the CPU, lesser the access time, and smaller the size. In a modern processor. such as Intel Xeon, which has a multi-core architecture, the L1 and L2 caches are private to each core while the L3 cache is shared among all the cores.

Information can leak in a number of ways from a cache memory. Let us take a brief look at the various ways in which information disclosure can occur in cache memory [6].

a) Premptive scheduling: Consider the case of a CPU with a single core in a virtualized environment. Assume the single core is allocated two VMs. Now, since the CPU can execute only one VM at any given time, it has to do periodic context switching between the two. During the switch, the scheduled VM has access to the state of the cache as left by the previous VM which leads to information leakage.

b) Hyper-Threading: In this technique two different threads can execute simultaneously on a single CPU core. This is achieved by allowing the two threads to share all of the CPU resources such as caceh, ALU etc. There is much more opportunity for information leakage in this case compared with the previous case.

c) Multicore: In a multicore architecture of CPU, each core has a private L1 and L2 cache. The L3 cache is shared among all the cores. This is a little restrictive environment for the attacker. The adversary has only one possibility, which is to probe the L3 cache to obtain information on the cache access pattern by the victim.

With this brief introduction to caches let us now look at the various types of side-channel attacks that have been found to be possible on cache memory.

II. Attack Classification

Cache based side-channel attacks fall under three broad categories:

- Access-driven Cache Attacks
- Time-driven Cache Attacks
- Trace-driven Cache Attacks

A. Access-driven Cache Attacks

A highly cited example for this type of attack is due to Percival [9]. Let us take an example of an adversary trying to recover the private key of AES encryption. The attacker and the encryption module are two different processes executing on the same CPU core. First, the attacker will try to fill the entire cache with his own data by loading a large array. He will then let the encryption process to execute. AES encryption makes use of lookup tables which in turn depend on the plain text and the key. Now, in order to do the encryption the appropriate table entries will be loaded into the cache. The loaded entries will replace some of the entries previously loaded by the attacker. After the encryption, the attacker will again load the entire array. He also measures the time taken to load the array into the cache lines. If the time taken to fill a cache line takes longer then he knows that the entry has been evicted by the encryption process and from that he can guess the index of the lookup table. Research has shown that it is possible to extract the entire 128 bits of the AES key using this attack [1], [8].

B. Time-driven Cache Attacks

This type of attack relies on measuring the execution time of a process for various inputs. For example, in case of AES encryption the total execution time of the process depends on the plain text and key. A common attack in this scenario is the "cache collision attack". This attack makes use of the experimental result that higher number of cache collision leads to less cache misses which in turn leads to shorter execution times [8], [3], [10]. By measuring the execution time for encryption of large number of plain texts using the same key the attacker identifies that one execution which has the lowest mean execution time. He then is able to deduce the entire AES key based on the last round attack [3], [10].

These attacks are classified further based on the position of the adversary:

- Passive time-driven cache attacks
- Active time-driven cache attacks

1) Passive time-driven cache attacks: In a passive attack the attacker cannot directly probe or modify the victim's cache. The passive attacker also does not have access to timing information on the victim's machine making the attack harder. The inability to measure accurately the time and lack of direct access to the cache creates noise in the attacker's measurements. In spite of this, the attack can be carried out using a large number of samples, such as the Bernstein's attack [2] to recover the AES key using $2^{27.5}$ measurements.

2) Active time-driven cache attacks: In an active attack, the attacker has direct access to victim's cache which enables him to cause collisions and even manipulate the state of the cache. The direct access also provides with the ability to measure the time accurately unlike the passive case. Osvik et al. [8] demonstrate an attack on AES that recovers the complete 128 bit key using 500,000 measurements which is much more efficient than Bernstein's passive attack.

C. Trace-driven Cache Attacks

This attack, like active time-driven attack requires the attacker to have direct access to victim's cache. Specifically, the attacker tries to observe the access pattern of cache lines by the victim and tries to manipulate or probe the cache lines based on the observation to his advantage. A commonly cited example of this type of attack is the Prime + Probe attack. As a first step, the attacker fills the cache lines with contents of certain memory address (Prime). After some time, the attacker tries to access the contents at the same memory address (Probe). If the time taken to access contents at the probe stage is longer than the prime stage then the attacker infers that the victim has accessed a pre-image (memory address) of the same cache line as the attacker. The leaked information here is the access record of the victim process.

Attacks based on this idea have been successfully demonstrated by researchers. The attacks are made easier due to the Hyper-Threading features available in processors where two threads share a number of hardware resources executing simultaneously. Percival [9] exploited this idea and demonstrated an attack on the RSA encryption where he was able to obtain the traces of modular exponentiation and multiplication operations of the algorithm. In another work, Neve [7] demonstrated the feasibility of trace-driven attacks on a single-threaded processor making it more severe and easier than the Hyper-Threading version. In yet another attack, Gullasch et al. [5] were able to extract the full key of AES encryption using Linux Scheduler.

The attacks describe in this section and before show the practical feasibility and the ease of conducting cache based side channel attacks.

III. Defense Mechanisms

Current mechanisms for defending against cache based side-channel attacks are either *hardware based* or *software based*. Hardware based solutions suggest a modification to existing cache design. This would involve a change in the chip manufacturing process. An example would be the work by Domnitser et al. [4] who propose a new cache design called Non-monopolizable caches (NoMo). The basic idea is to reserve a few cache lines for each process non-evictable. The authors claim that their approach requires only a slight modification to the hardware's cache replacement algorithm with no other support.

Software based approaches, on the other hand, work by modification to operating system kernel without requiring any changes to underlying hardware. Kim [6] et al. present an idea of stealth memory where each virtual machine or a process is allocated certain parts of cache lines that are never evicted by another process. This non-evictable cache memory is called stealth memory. Since a process or VM cannot evict the locked cache lines side channel attacks can be mitigated. The solution is an extension of hypervisor code in the case of VMs or operating system code in the case of single machine(s).

The final term paper will include a detailed overview of all the software and hardware based defense mechanisms proposed in the research literature.

IV. Proposed Outline for Final Paper

In the full version of my term paper I plan to discuss the following:

- Evolution of cache based side-channel attacks in the research literature
- Comprehensive analaysis of various defense mechanisms – pros and cons
- My own critique on the nature of the problem and proposed solutions
- Finally, expected trajectory of the problem that future solutions should consider

References

[1] Onur Aciiçmez. Yet another microarchitectural attack:: exploiting i-cache. In *Proceedings of the 2007 ACM workshop on Computer security architecture*, pages 11–18. ACM, 2007.

[2] Daniel J Bernstein. Cache-timing attacks on aes, 2005.

[3] Joseph Bonneau and Ilya Mironov. Cache-collision timing attacks against aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 201–215. Springer, 2006.

[4] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):35, 2012.

[5] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games–bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy*, pages 490–505. IEEE, 2011.

[6] Taesoo Kim, Marcus Peinado, and Gloria Mainar-Ruiz. Stealthmem: system-level protection against cache-based side channel attacks in the cloud. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 189–204, 2012.

[7] Michael Neve and Jean-Pierre Seifert. Advances on access-driven cache attacks on aes. In *International Workshop on Selected Areas in Cryptography*, pages 147–162. Springer, 2006.

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

[9] Colin Percival. Cache missing for fun and profit, 2005.

[10] Kris Tiri, Onur Acıiçmez, Michael Neve, and Flemming Andersen. An analytical model for time-driven cache attacks. In *International Workshop on Fast Software Encryption*, pages 399–413. Springer, 2007.