VET

CACHE ATTACK IN JAVASCRIPT

Authors :
Pierre RIBAULT

Entity :
ISTIC, Université Rennes 1

Introduction

Cryptographic algorithms that are secure against known theoretical attacks can still be vulnerable to side-channel attacks because of the flaws in their implementations (P.C.Kocher, s.d.). Side-channel attack exploit the fact that micro-architectural resources, like temperature, time to access etc.

A particular attack is the one using caches, it exploits the fact that it is shared between processes.
We will see in the text that it is possible to perform this attack remotely from a web browser.

BACKGROUND AND STATE OF ART

Memory Hierarchy:

In an information system there are several memories, the slowest ones that are the least expensive in quantity, they are far from the processor and allow the backup of large data while the fastest ones are near the processor.

To hide the latency time of memory accesses, CPUs use different levels of cache.

They are often three, variable in size and speed, the smallest (L1) cache is the fastest and the smallest, the largest and slowest cache is the L3 cache, which is directly connected to RAM, this is the last level cache.

When the CPU wants to access data in memory, it first checks whether the corresponding memory address is, available in its cache, if it is available it is a hit and the access time is fast.

If the address is not available in the cache, the CPU will retrieve the information from the RAM and replace an element of the cache with new information, increasing latency, this event is referred to as "miss".

Intel's microarchitecture is different from that of AMD, the first one is inclusive, if an element exists in the L1 level then it exists in the other levels.

The ejection of a line in the last level is also ejected from caches L1 and L2.

The second is exclusive; the different parts of this report are based on the first architecture.

COVERT CHANNEL

Cache attacks are based on the timing difference between cache and non-cache. They can be used to build side-channel attacks and covert channels.

There are two types of attacks, those based on shared memory that are not applicable in all environments where shared memory is not available, for example, because they are prohibited for security reasons. The second type of access-based attacks, called Prime+Probe is not based on shared memory and therefore applicable in protected environments. This attack has a lower granularity, and is more prone to noise than other attacks.
The covert channels on the last cache level allow two processes that are not allowed to communicate, or that there is no power communication channel even if they do not run on the same CPU core.

APPLICATION WITH JAVASCRIPT

The paper “The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications” presented how a side-channel cache attack can be launched from a untrusted webpage, the authors explain how to implement a side-channel cache attack in JavaScript for spying the user’s behaviour. The PRIME+PROBE attack follows four steps according to the paper:

- Creating an eviction set for one or more relevant cache sets
- Priming the cache set
- Triggering the victim operation
- Probing the cache set again

JAVASCRIPT

JavaScript is dynamically typed language evaluated on the client side, by default all web browsers execute the JavaScript code found in web pages.

For security reasons, the JavaScript code runs in a sandbox environment, and has system access restrictions .There are no pointer notions in JavaScript, so we can not know the virtual addresses of variables.

CREATING AN EVICTION SET

In the paper, an eviction set is defined by “sequence of variables (data) that are all mapped by the CPU into a cache set that is also used by the victim process”. They managed to optimize a brute force attack with two optimizations. The basic attack is:

- Define an arbitrary victim address in the memory.
- To find by brute force the possible addresses among those were the expulsion stamp are in the same cache as the first one.

This method is too slow and make the attack unnecessary.

To solve this problem they created the algorithm [1].

Bypass Protection

Research has shown that micro-architectural attacks are possible through web sites using JavaScript (Oren, et al., October 2015.).

The JavaScript function "performance. Now" gave a resolution to the nanosecond. That is why W3C consortium and web browsers have modified JavaScript timers to render these attacks inoperative by rounding the timer to 5 nanoseconds, which would distort the measurement of access time.

The article "Fantastic Timers and Where to Find Them: High-Resolution Micro-architectural Attacks in JavaScript" (Schwarz, et al., 2017)demonstrates how to bypass the defences by creating new timers.

This makes all existing attacks possible, and it demonstrates how to create a covert DRAM-based channel between a website and an unprivileged application in a virtual machine.

This part show in first how to retrieve a high-precision timer and the next part how to get a new timer. This one is unusable because it is not precise enough, but shows the possibility of finding timers in functions that are not intended.

Retrieving a high-precision timer

The article points out "as the source of the underlying clock has a high resolution, the difference between two clock edges only varies to the extent that the underlying clock". This method allowed them to move from a resolution of 5µs to 500 ns (on Firefox v. 51 and Chrome v. 53). So that we can retrieve a precise timestamp because the time between two edges is constant which gives us a basis to build a clock. To use this technique, we must first calibrate to do this by increasing a counter between two clock edges in a waiting loop, this will give us the number of steps that can be used for interpolation.
The resolution we can retrieve is the time it took to increase the counter.
To use this timer, we expect a clock front to start the operation we want to time, and then at the end of the operation we wait for the next clock front by increasing the counter.
We get the number of increments, we multiply it by the time it takes to make an entire clock face that gives us the interpolated time.

Other possibilities

This paper also describes how to create synchronization sources. The one I will describe is based on CSS animations, and is independent of JavaScript. By defining an animation, that changes the size of an element from 0px to 1,000,000 px in one second.
Web browsers limit CSS animations to 60 fps. So the animation is divided into 60 frames per second, which gives a resolution of 1/60 = 16 ms.

To retrieve the timestamp we just need to retrieve the current width of the element using the JavaScript function “windows. getComputedStyle (element, null). getPropertyValue ("width")”.

Error in cache Covert Channel

This part of the text is divided into two parts. The first one will highlight errors in the on cache covert channel and the second how to build a robust error-handling protocol. The source of this part is the article "Maurice et al. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. NDSS 2017" (Maurice, et al., March 2017).

Causes of Errors

A covert channel consists of two entities, a transmitter and a receiver; the only way for them to communicate is through the same covert channel. The transmitter sends bits with or without expelling lines from the cache.

The receiver reads the bits by constantly probing a set in its cache. The type of bits received depends on the access time to a line.

This measured time, perhaps influenced by outside activities. The transmitter and receiver can also be ejected from the cache at the same time, this case create substitution errors. In addition, the transmitter can not determine how many cycles it took the receiving process to read a transmitted bit. Neither process is synchronized and they have no way of informing the other process of its status (this case creates insertion or deletion errors on the receiver side).

Intensive use of the cache can create interference, if a process is carried out to remove the receiver line, a' 0' is changed to' 1', conversely if the transmitter is unable to open its cache, the transmitter receives a' 0' instead of a' 1'.

Insertion and deletion errors are due to synchronization errors,

If the receiver probes it's cache while the transmitter does not send anything, it receives '0' and creates an insertion error. When the receiver is interrupted while the transmitter is sending bits, it cannot read the transmitted bits, so this creates deletion errors.

How to build a robust protocol for covert channel

The article describes a communication protocol, developed for covert channels.

For the data-link layer:

- Fill in a buffer with the data you want to send.
- We divide the data into parts of the same size, which we improve with an error correction, which gives us a packet.
- The packets are divided into small, easily transferable words between scheduling.
- These words are again encoded to protect against synchronization errors.

On the physical layer to transmit a word:

The transmitter triggers the last level cache memory sufficiently for the receiver lines to be expelled from the L1 cache memory.

The number of lines expelled from the receiver's L1 cache can be used to infer the transmitted symbol.

The sender can not tell which lines of the last level cache are present in the receiver's L1 cache but "In practice, we have found that the optimal number of lines initiated by the sender is the number of lines in the last level cache".

For reception, the receiver probes its lines, the greater the number of lines, the more likely it is to cause interferences, but the greater the number of bits per symbol it has.

CONCLUSION

Since 2015 with the publication of Oren (Oren, et al., October 2015.), the power of hidden channel attacks has increased, in fact the ability to run them remotely with web pages.

The various attempts to find a solution to these attacks did not work, the publication of Maurice (Maurice, et al., March 2017)demonstrates the possibility of finding timers in several ways, some of which by distorting the purpose of using certain processes (e. g. animations) to obtain timers precise enough to render the mitigations made by web browsers obsolete.

One can imagine that by the difficulty to counteract these attacks, and therefore the need to remain informed on developments in this field.

Bibliography

Maurice, C. et al., March 2017. *the Other Side: SSH over Robust Cache Covert Channels in the Cloud,* s.l.: s.n.

Oren, Y., Kemerlis, V., Sethumadhavan,, S. & Keromytis, A., October 2015.. *The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications,* s.l.: s.n.

P.C.Kocher, n.d. Differential power analysis. *CRYPTO, ser. Lecture Notes in Computer Science,* Volume 1666, pp. 388-397.

Schwarz, M., Maurice, C., Gruss, D. & Mangard, S., 2017. *Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript,* s.l.: s.n.

Algorithm 1:

Let S be the set of currently unmapped page-aligned addresses, and address x be an arbitrary page-aligned address in memory.

1. Repeat k times:

    (a) Iteratively access all members of S.

    (b) Measure t1, the time it takes to access x.

    (c) Select a random page s from S and remove it.

    (d) Iteratively access all members of S\s.

    (e) Measure t2, the time it takes to access x.

    (f) If removing s caused the memory access to speed up considerably (i.e., t1 − t2 > *thres*), then this address is part of the same set as x. Place it back into S.

    (g) If removing s did not cause memory access to speed up considerably, then s is not part of the same set as x.

2. If |S| = 12, return S. Otherwise report failure.