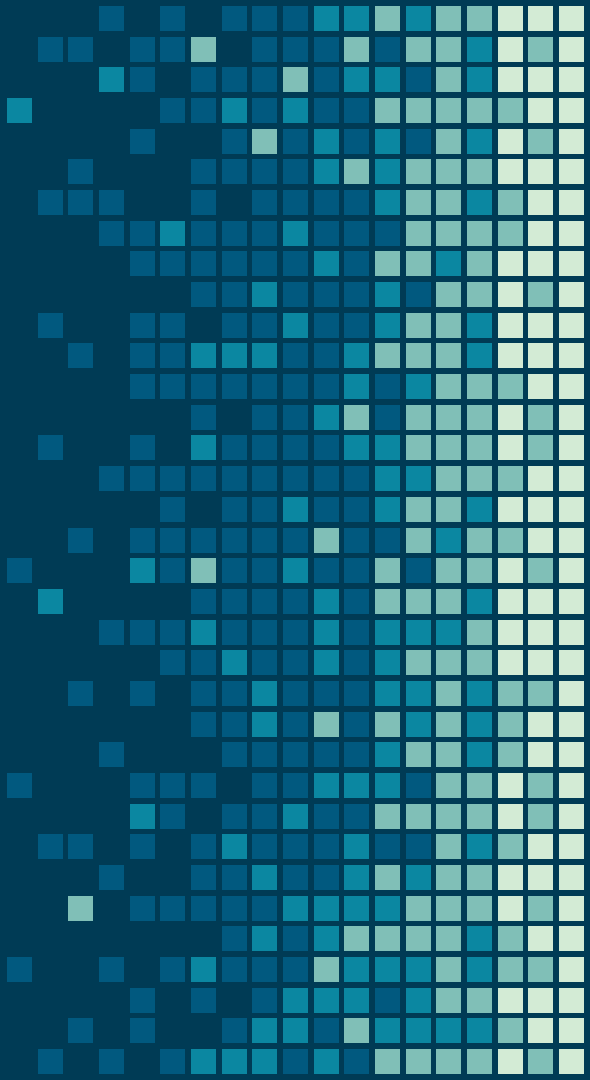
CACHE ATTACKS AND THEIR APPLICATIONS

Callemard Alexis, Nogues Maël, Ribault Pierre, Zemmouri Iskander



Introduction

The classical way of doing things

- Step 1: somehow convince a target to visit the attacker's malicious website.
- Step 2: involves some sort of browser vulnerability to exploit, one way or another.

What we want instead

- Same benefits as with the older way
- Independence from software bugs
- Longevity of the exploit

The newer way of doing things:

- Same benefits when it comes to spying
- But it’s not a flaw, it’s hardware design!

Meet cache attacks

- Step 1: still convincing the target to visit the attacker's malicious website
- Step 2: monitoring the CPU cache to snoop on the target

One step beyond: covert channels

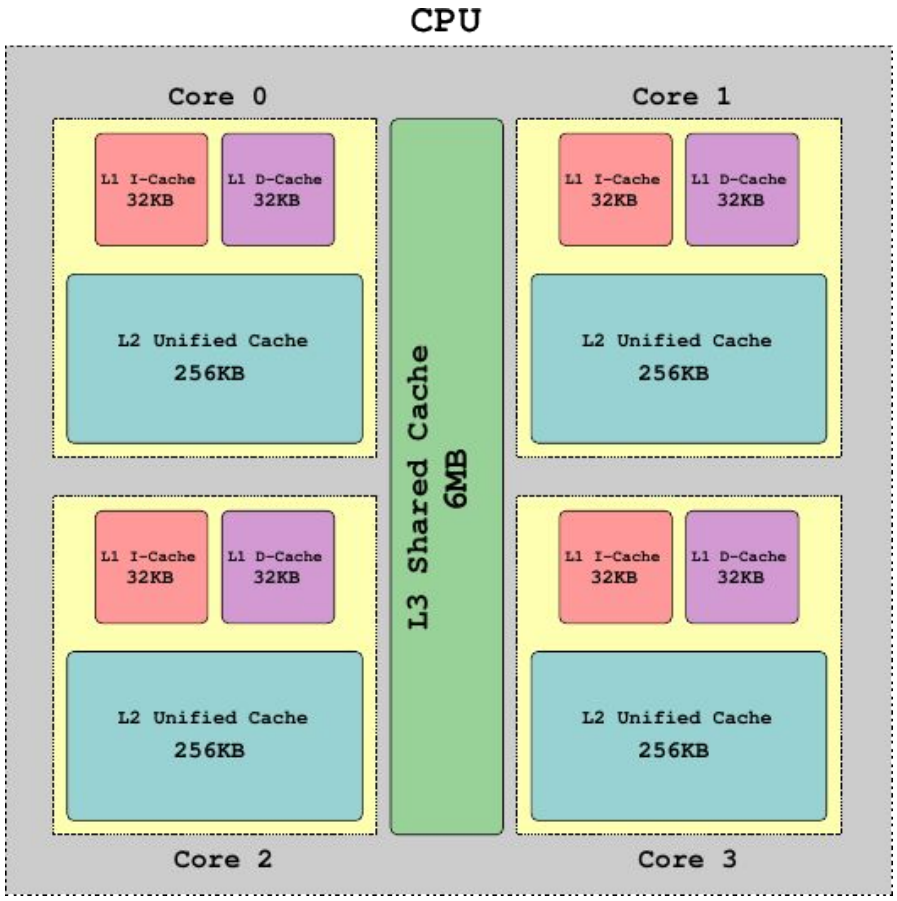
Covert channels are simply unauthorized and unexpected channels of communication.

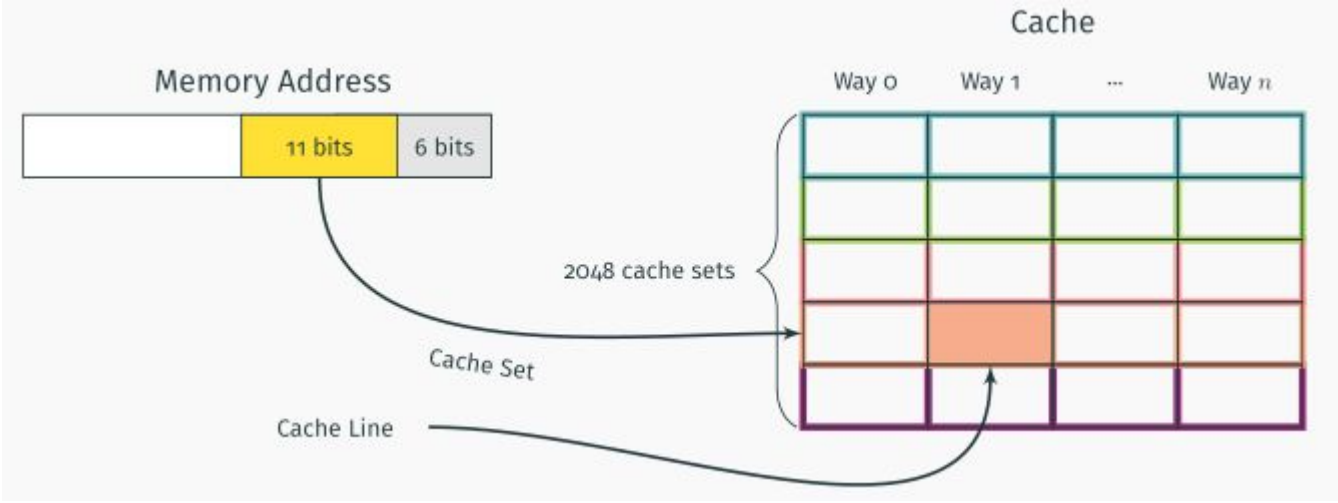
Covert channels based on cache attacks can be made cross-vm, thus even defeating virtual machines' seemingly perfect isolation.

Cache attacks are a type of side-channel attacks.

A side-channel is basically any unaccounted for or disregarded leak of information.

Cache attacks obviously involve memory caches, or more precisely, monitoring memory caches


CPU
Core 0
L1 I-Cache
32KB
L1 D-Cache
32KB
L2 Unified Cache
256KB
Core 1
L1 I-Cache
32KB
L1 D-Cache
32KB
L2 Unified Cache
256KB
L3 Shared Cache
6MB
Core 2
L1 I-Cache
32KB
L1 D-Cache
32KB
L2 Unified Cache
256KB
Core 3
L1 I-Cache
32KB
L1 D-Cache
32KB
L2 Unified Cache
256KB


Cache
Memory Address
Way 0
Way 1
...
Way n
11 bits
6 bits
2048 cache sets
Cache Set
Cache Line

Summary

1. Cache-attacks: a JavaScript example
2. Engineering alternative timing sources
3. Errors in side-channels and how to fix them

1. Cache-attacks: a JavaScript example

Motivations

- Scalability.
- No physical access needed.
- Allows for profiling of basic users.
  - On social media.
  - On banking websites.

Prime+Probe

- Makes use of access time difference.
- Applied to one cache set.
- Works across CPU cores since the LLC is shared.

Prime+Probe

- Creating an eviction set for one or more relevant cache sets.
- Priming (filling) the cache set.
- Triggering the victim’s operation.
- Probing the cache set again.

Creating an eviction set

- Variables mapped by the CPU to a set used by the victim.
- Fix an arbitrary address and brute force.
  - Optimisation 1: shrink the set by randomly removing elements.
  - Optimisation 2: if physical addresses P1and P2 share a cache set, then for any value of Δ, $P1 \oplus \Delta$ and $P2 \oplus \Delta$ also share a cache set.

Priming and probing

- Replace each entry in the CPU cache with the eviction set.
- Measure Probe time precisely.
- Linked-list (ensures access before measurement).
- Randomly permuting elements (stride prefetching).
- Access from alternating directions (avoid too many cache misses).

Identifying interesting cache regions

- Correlate cache sets to code or data belonging to the victim.
- Machine learning
  - Derive meaning from cache set latency measurements.
  - Incentivize the victim to perform an action.

2. Timers and how to find them

Timers and how to find them

- W3C and browsers vendors eliminated fine-grained timers from JavaScript.
  - Wrong solution.
  - Other ways of finding / creating timers.

Recovering a high-resolution timer

- Observing clock edges.
  - Clock edges : time at which the timestamp is an exact multiple of its resolution.
  - Increment a counter between clock edges to get an higher resolution.
  - Attacks:
    - Clock interpolation.
    - Edge thresholding.

Clock interpolation

- Usage:
  - Busy wait for a clock edge.
  - Start the operation to time.
  - Busy wait for the next clock edge, incrementing the counter.
- Results: from a 100ms timer to a 15µs timer.

Alternative timing primitives

- Timeouts:
  - Use the timer of the browser with the setTimeout function.
  - Concurrent timer-based callback simulate a counting thread by incrementing a variable.
  - Microsoft browsers have another function called setImmediate that allows a resolution of up to 50µs.

Alternative timing primitives

- Other methods:
  - Message passing.
  - Message channel.
  - CSS animations.
  - SharedArrayBuffer.

Results of different timing primitives

	Free-running	Firefox 51	Chrome 53	Edge 38	Tor 6.0.4	Fuzzyfox
performance.now	✓	5 µs	5 µs	1 µs	100 ms	100 ms
CSS animations	✓	16 ms	16 ms	16 ms	16 ms	125 ms
setTimeout		4 ms	4 ms	2 ms	4 ms	100 ms
setImmediate		–	–	50 µs	–	–
postMessage		45 µs	35 µs	40 µs	40 µs	47 ms
Sub worker	✓	20 µs	–[2]	50 µs	15 µs	–
Broadcast Channel	✓	145 µs	–	–	55 µs	760 µs
MessageChannel		12 µs	55 µs	20 µs	20 µs	45 ms
MessageChannel (W)	✓	75 µs	100 µs	20 µs	30 µs	1120 µs
SharedArrayBuffer	✓	2 ns[3]	15 ns[4]	–	–	2 ns[3]
Interpolation[1]		500 ns	500 ns	350 ns	15 µs	–
Edge thresholding[1]		2 ns	15 ns	10 ns	2 ns	–

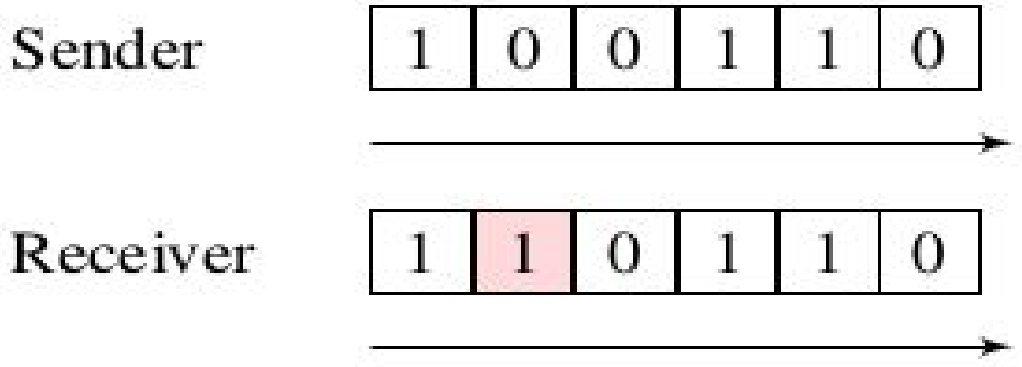
3. Errors in side-channels and how to fix them

Errors in side-channels

- Noise
- Descheduled sender/receiver

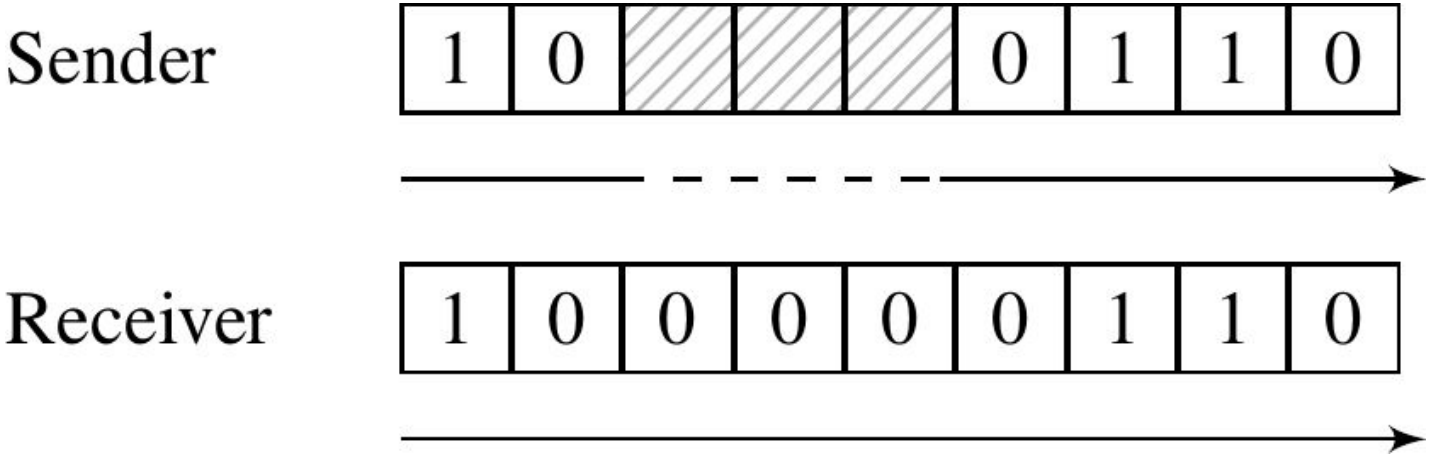
Errors in side-channels

- Noise cause a substitution errors.



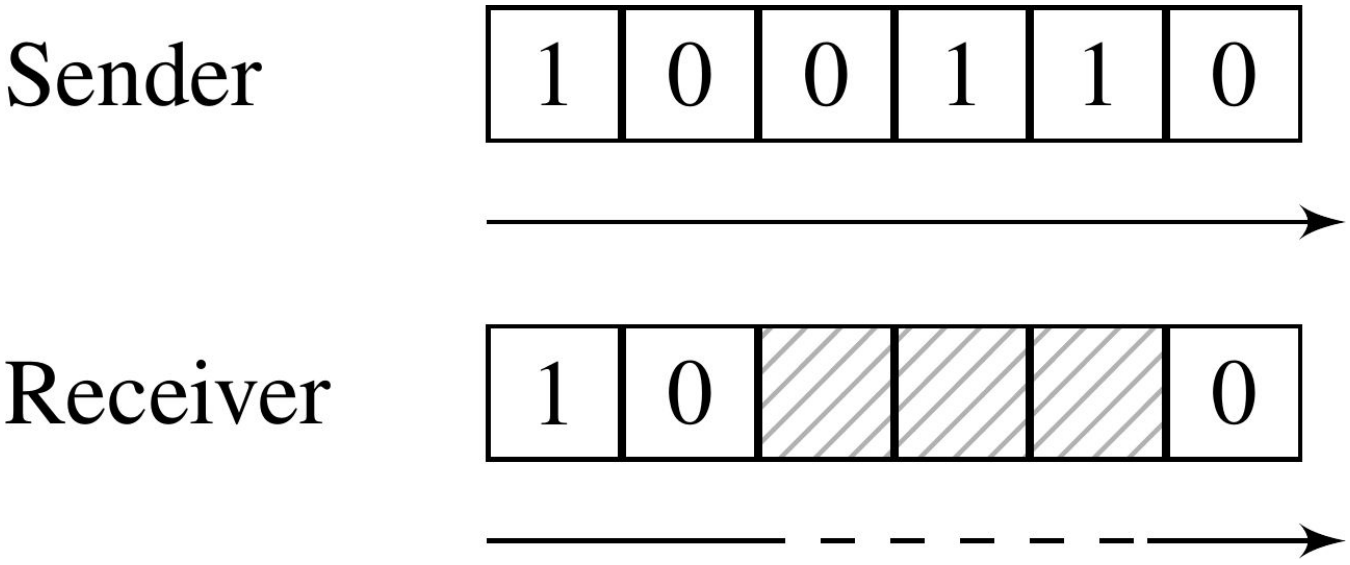
Errors in side-channel

- Sender descheduled create insertion error.



Errors in side-channel

- Receiver descheduled create deletion error.

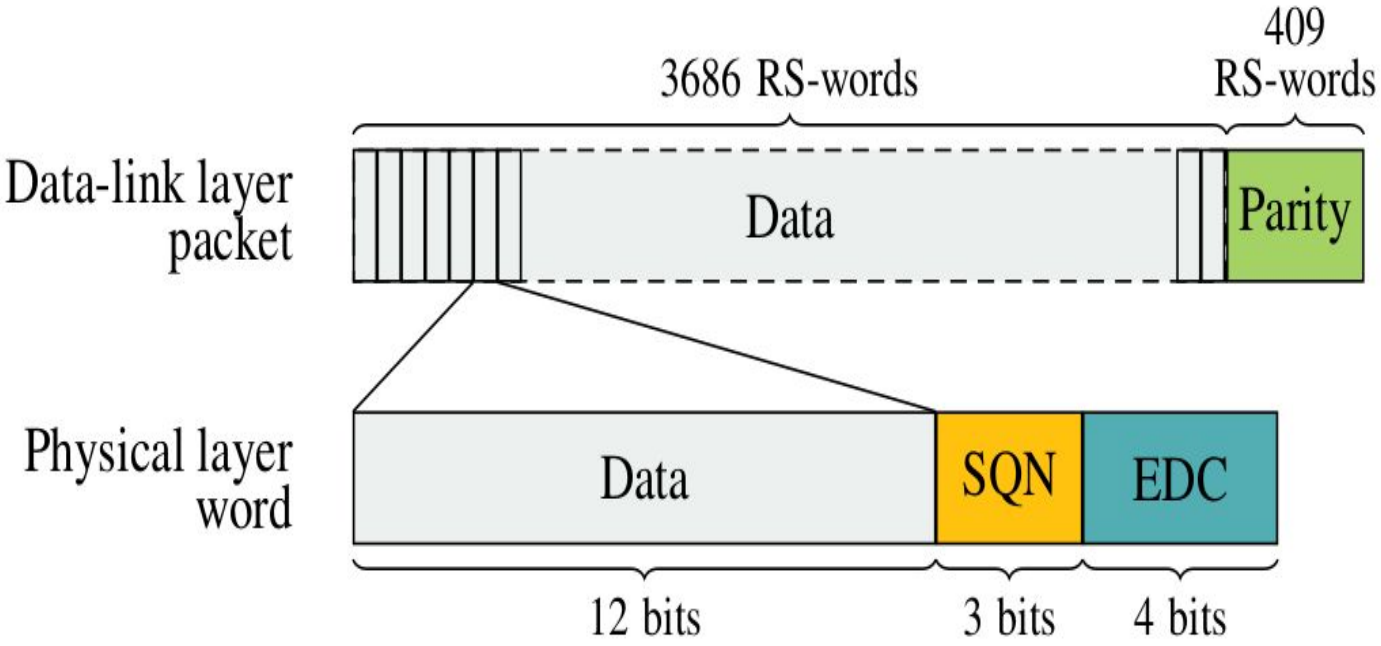


Robust covert-channels

Communication protocol.

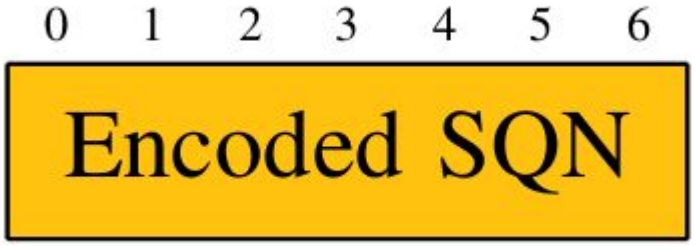
- Physical layer.
- Data-link layer.

Robust cache covert-channels

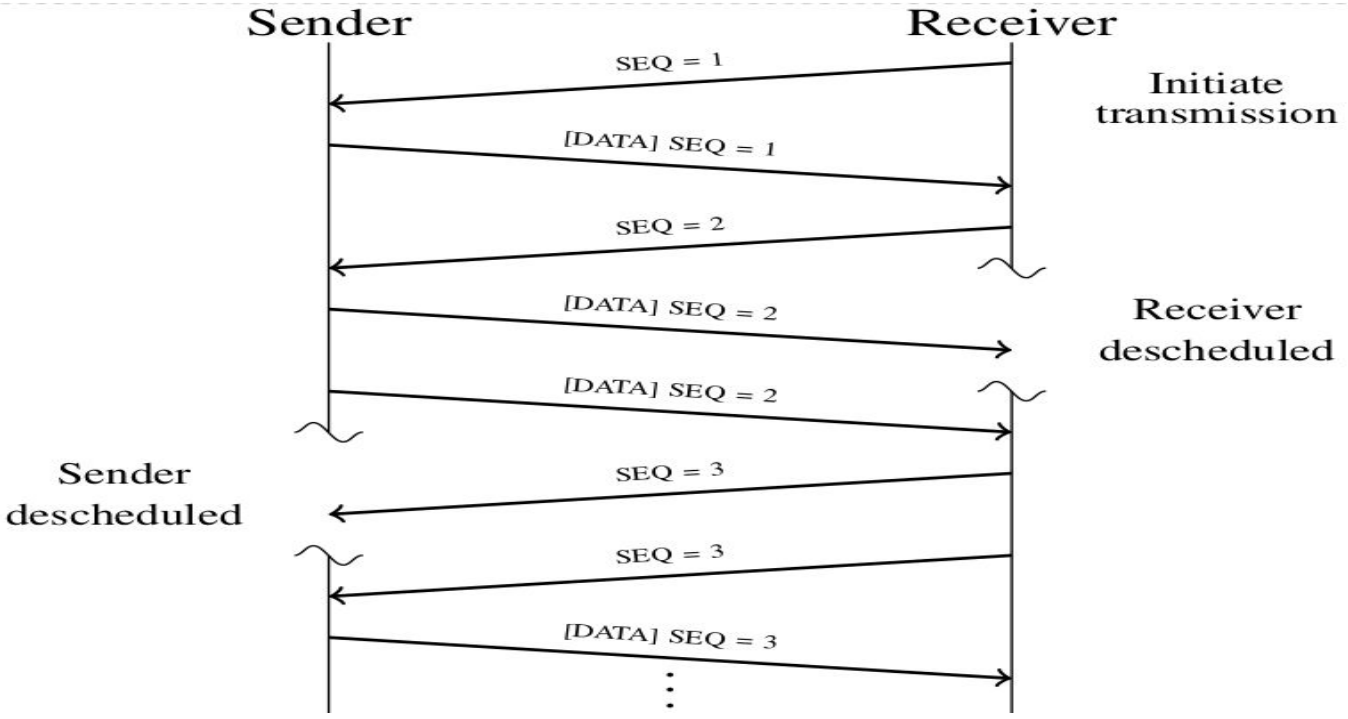


Robust cache covert-channels

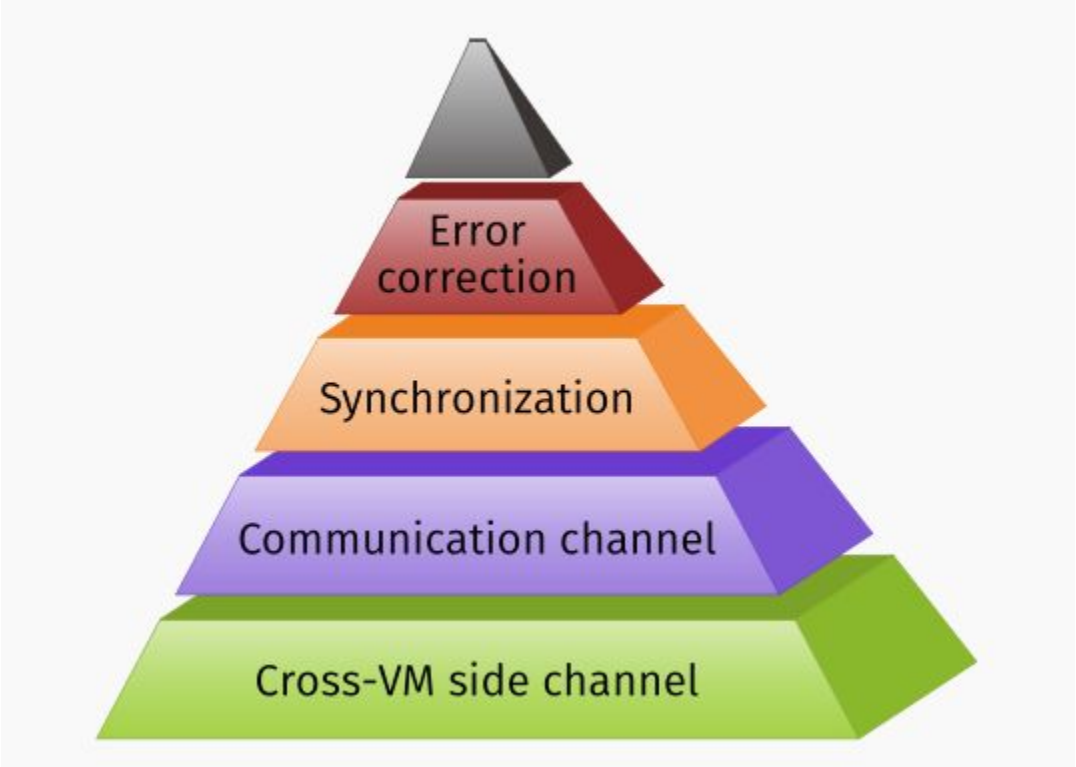
Substitution errors apply

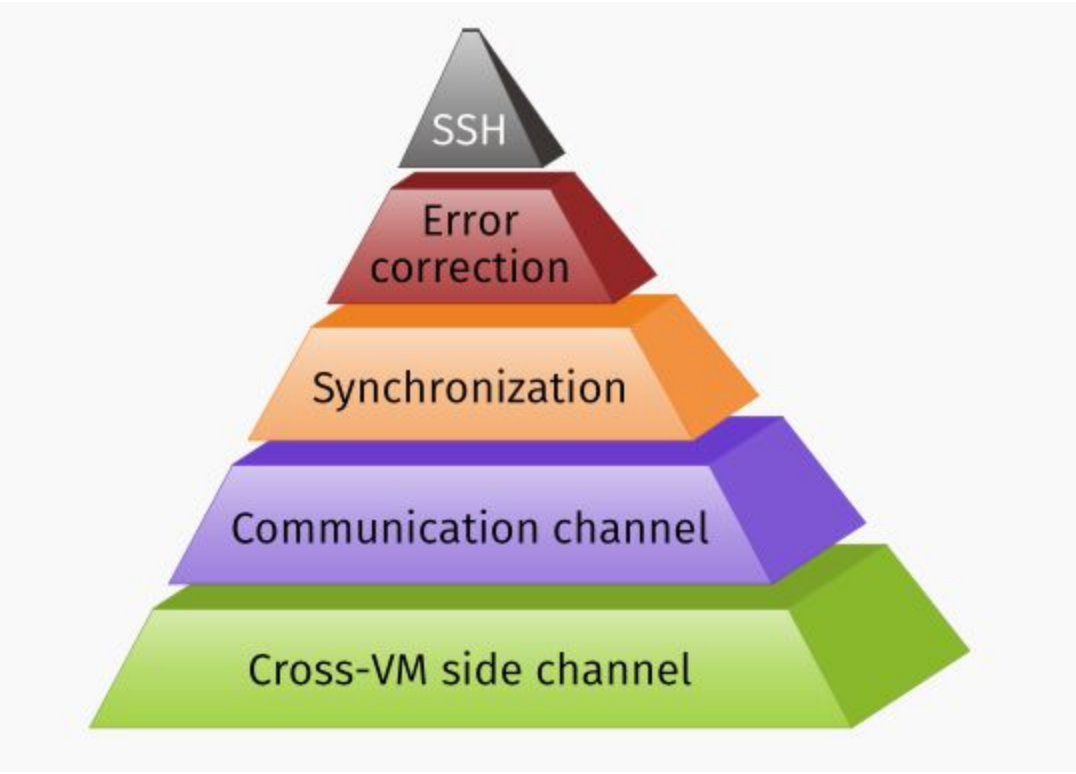


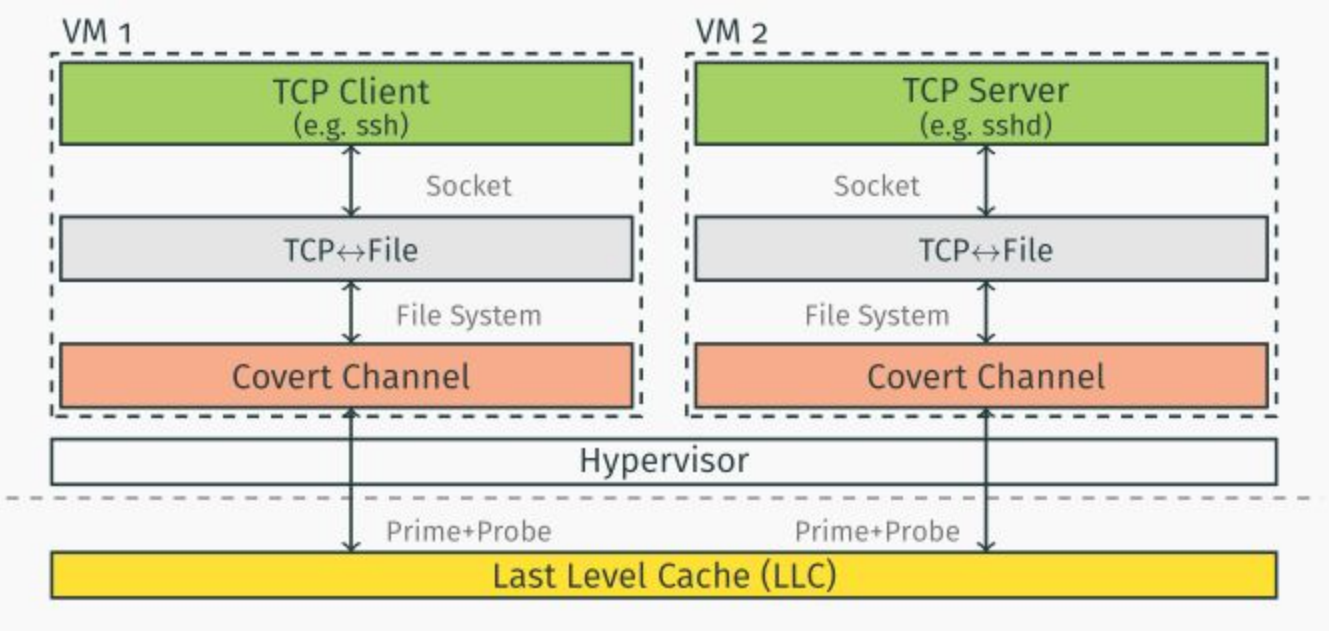
Deletion errors



Insertion error


Error correction
Synchronization
Communication channel
Cross-VM side channel


SSH
Error correction
Synchronization
Communication channel
Cross-VM side channel


VM 1
TCP Client
(e.g. ssh)
Socket
TCP↔File
File System
Covert Channel
VM 2
TCP Server
(e.g. sshd)
Socket
TCP↔File
File System
Covert Channel
Hypervisor
Prime+Probe
Prime+Probe
Last Level Cache (LLC)

Conclusion

- Cache attacks are practical
- Cache-based covert-channels are also practical
- The covert channel can be made noise-free, thus reliable
- Noise is no protection against cache attacks
- Removing timer APIs is also no protection against cache attacks
- Promising attacks that may develop and yield creative stuff in the future

Sources

1. Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. The spy in the sandbox: Practical cache attacks in javascript and their implications. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015, pages 1406–1418. ACM, 2015.

2. Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay R ̈omer. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society, February 2017.

3. Michael Schwarz, Cl´ementine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript . In Proceedings of the 21st International Conference on Financial Cryptography and Data Security (FC'17), FC, April 2017.

Questions?