ARMageddon

How Your Smartphone CPU Breaks Software-Level Security And Privacy

Moritz Lipp and Clémentine Maurice

November 3, 2016—Black Hat Europe

Motivation

- Safe software infrastructure does not mean safe execution

Motivation

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware

Motivation

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- We focus on the CPU cache

Motivation

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- We focus on the CPU cache
- Cache attacks can be used for covert communications and attack crypto implementations

Motivation

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- We focus on the CPU cache
- Cache attacks can be used for covert communications and attack crypto implementations
- Only been demonstrated on Intel x86 for now

Motivation

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- We focus on the CPU cache
- Cache attacks can be used for covert communications and attack crypto implementations
- Only been demonstrated on Intel x86 for now
- But why not on ARM?

Who We Are

Who We Are

- **Moritz Lipp**
- Master Student, Graz University Of Technology
- 🐦 @mlqxyz
- ✉ mail@mlq.me

Who We Are

- **Clémentine Maurice**
- PhD in InfoSec; Postdoc, Graz University Of Technology
- 🐦 @BloodyTangerine
- ✉ clementine.maurice@iaik.tugraz.at

The rest of the team

The rest of the research team

- Daniel Gruss
- Raphael Spreitzer
- Stefan Mangard

From Graz University of Technology

Demo

Outline

- Background information

Outline

- Background information
- What are the challenges for cache attacks on ARM?

Outline

- Background information
- What are the challenges for cache attacks on ARM?
- How to solve those challenges

Outline

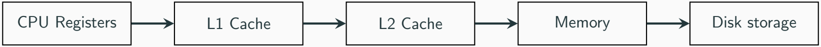
- Background information
- What are the challenges for cache attacks on ARM?
- How to solve those challenges
- Attack scenarios

Outline

- Background information
- What are the challenges for cache attacks on ARM?
- How to solve those challenges
- Attack scenarios
- Tools

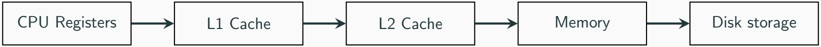
Cache Attacks

Memory Hierarchy



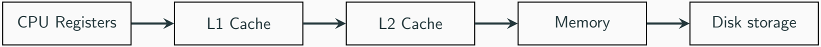
- Data can reside in

Memory Hierarchy



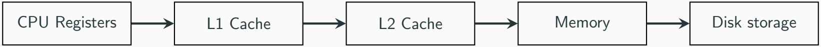
- Data can reside in
    - CPU registers

Memory Hierarchy



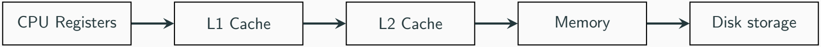
- Data can reside in
    - CPU registers
    - Different levels of the CPU cache

Memory Hierarchy



- Data can reside in
    - CPU registers
    - Different levels of the CPU cache
    - Main memory

Memory Hierarchy



- Data can reside in
    - CPU registers
    - Different levels of the CPU cache
    - Main memory
    - Disk storage

Cache Attacks

- Exploit timing differences of memory accesses:

Cache Attacks

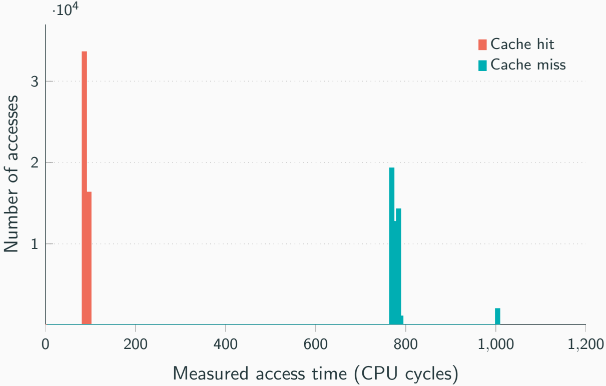
- Exploit timing differences of memory accesses:
  - cache → fast (cache hit)

Cache Attacks

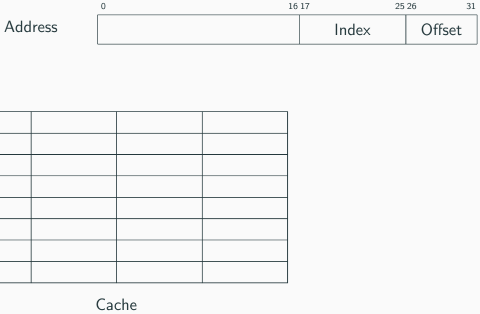
- Exploit timing differences of memory accesses:
  - cache → fast (cache hit)
  - main memory → slow (cache miss)

Cache Attacks

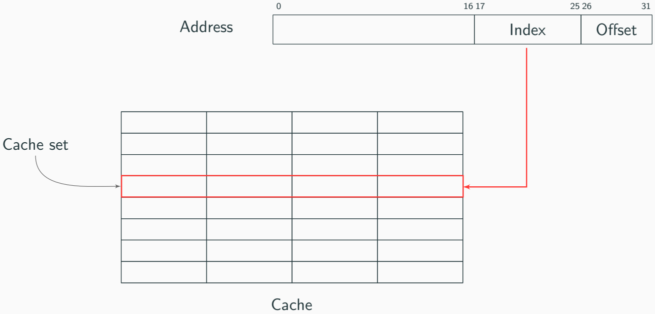
- Exploit timing differences of memory accesses:
  - cache → fast (cache hit)
  - main memory → slow (cache miss)



Set-Associative Caches

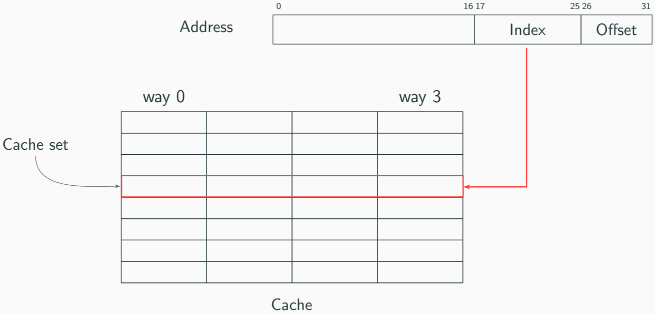


Set-Associative Caches



Data loaded in a specific set depending on its address

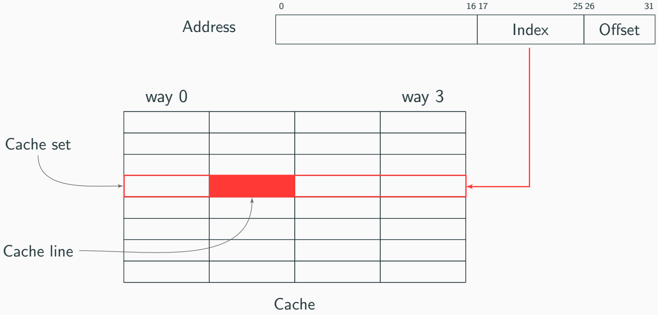
Set-Associative Caches



Data loaded in a specific set depending on its address

Several ways per set

Set-Associative Caches

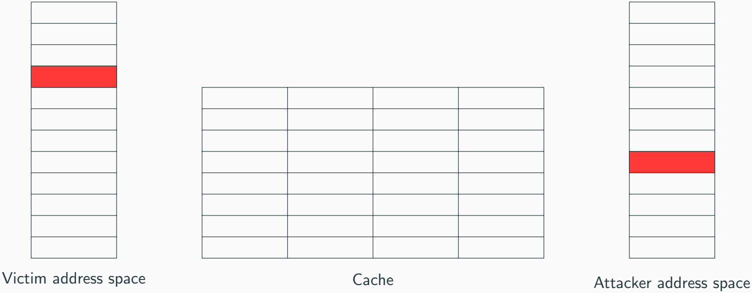


Data loaded in a specific set depending on its address

Several ways per set

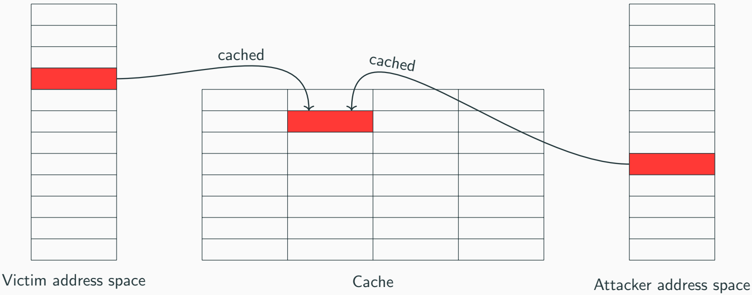
Cache line loaded in a specific way depending on the replacement policy

Cache Attacks: Flush+Reload



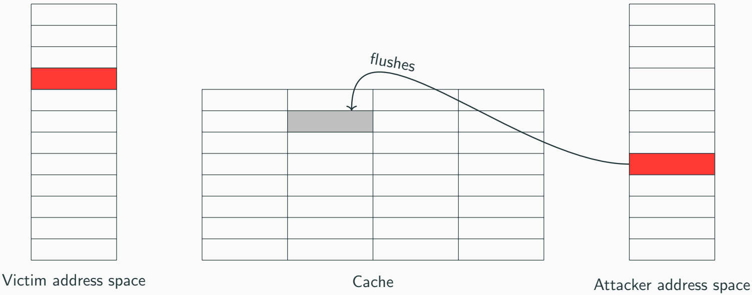
Step 1: Attacker maps shared library (shared memory, in cache)

Cache Attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

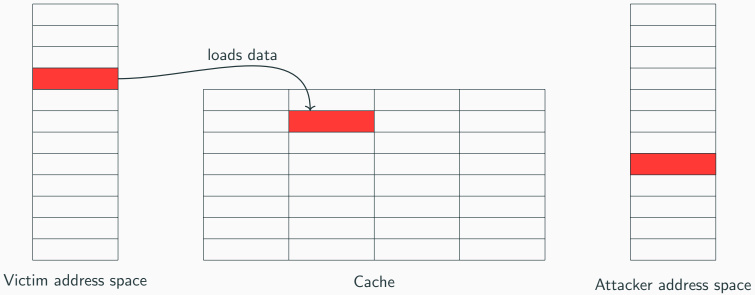
Cache Attacks: Flush+Reload



Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Cache Attacks: Flush+Reload

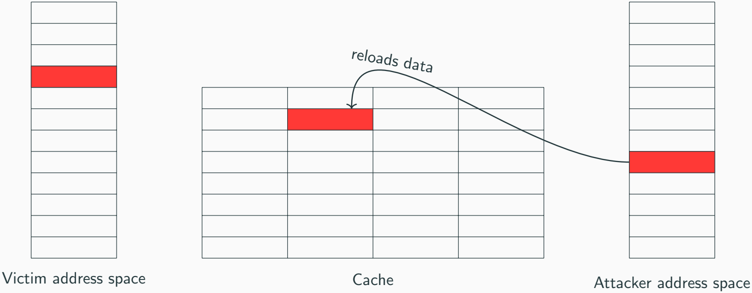


Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

Step 3: Victim loads the data

Cache Attacks: Flush+Reload



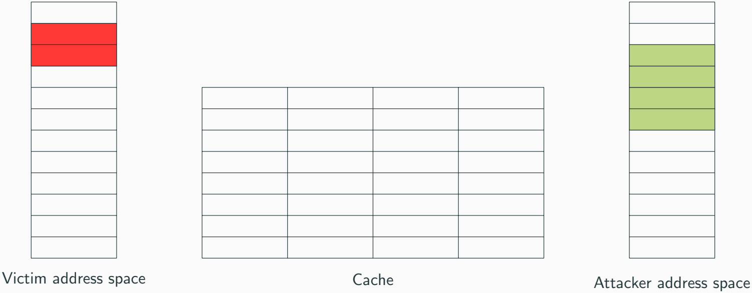
Step 1: Attacker maps shared library (shared memory, in cache)

Step 2: Attacker flushes the shared cache line

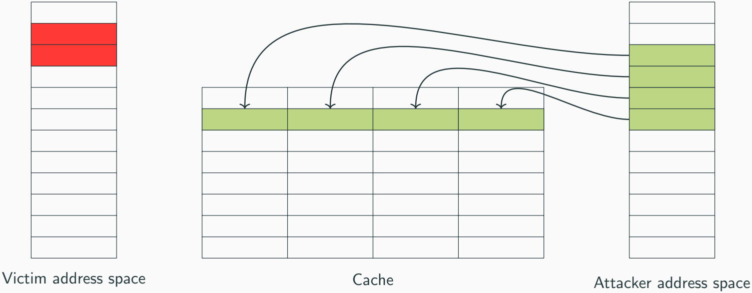
Step 3: Victim loads the data

Step 4: Attacker reloads the data

Cache Attacks: Prime+Probe

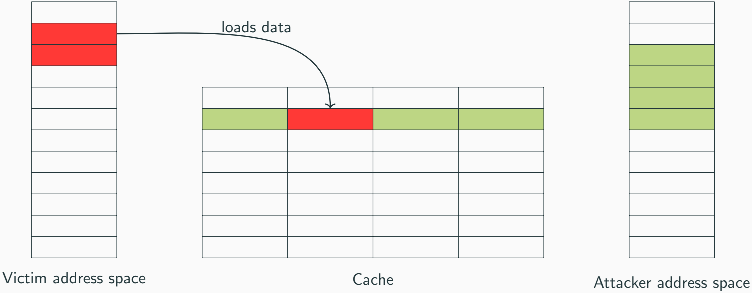


Cache Attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

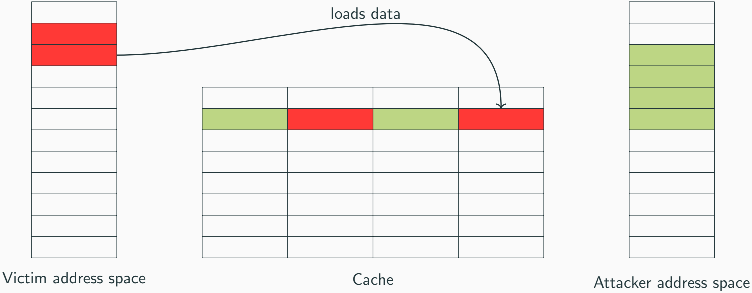
Cache Attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

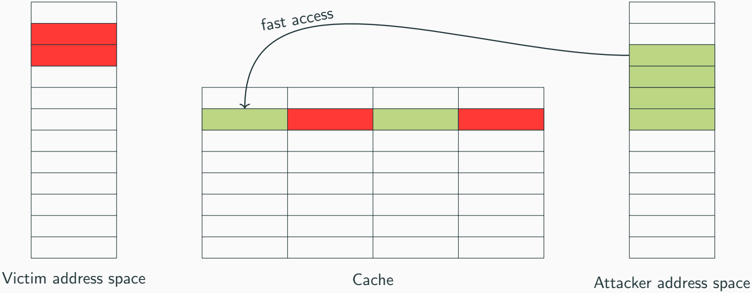
Cache Attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Cache Attacks: Prime+Probe

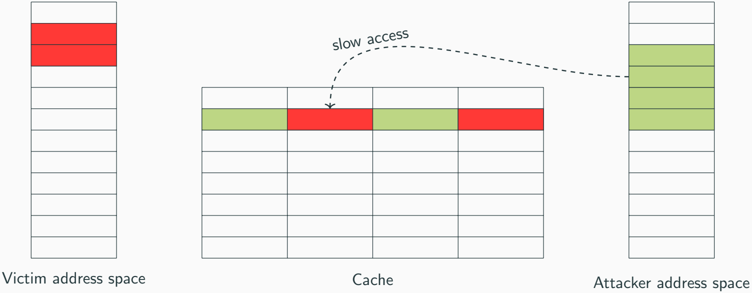


Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Cache Attacks: Prime+Probe



Step 1: Attacker primes, *i.e.*, fills, the cache (no shared memory)

Step 2: Victim evicts cache lines while running

Step 3: Attacker probes data to determine if set has been accessed

Differences between Intel x86 and ARM

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines
- Cache maintenance instructions

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines
- Cache maintenance instructions
    - Intel x86: Unprivileged `clflush` instruction

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines
- Cache maintenance instructions
    - Intel x86: Unprivileged `clflush` instruction
    - ARMv7-A: Only privileged cache maintenance instructions

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines
- Cache maintenance instructions
  - Intel x86: Unprivileged `clflush` instruction
  - ARMv7-A: Only privileged cache maintenance instructions
  - ARMv8-A: Privileged instructions can be unlocked for userspace

Cache maintenance

- Basic operation for cache attacks: invalidate cache lines
- Cache maintenance instructions
  - Intel x86: Unprivileged `clflush` instruction
  - ARMv7-A: Only privileged cache maintenance instructions
  - ARMv8-A: Privileged instructions can be unlocked for userspace

Challenge #1

No flush instruction

Cache eviction

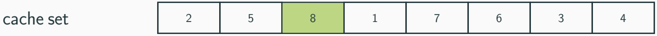
- Fill the whole cache

Cache eviction

- Fill the whole cache → too slow

Cache eviction

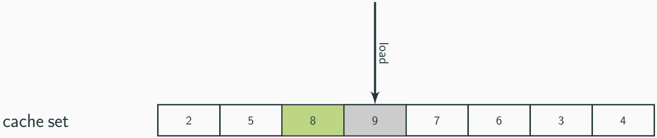
- Fill the whole cache → too slow
- Fill a specific cache set



cache set	2	5	8	1	7	6	3	4

Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set

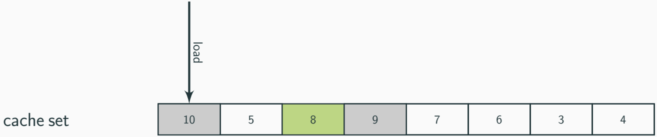


load

cache set	2	5	8	9	7	6	3	4

Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set

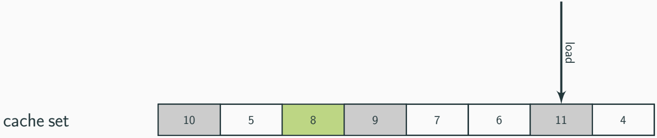


load

cache set	10	5	8	9	7	6	3	4

Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set

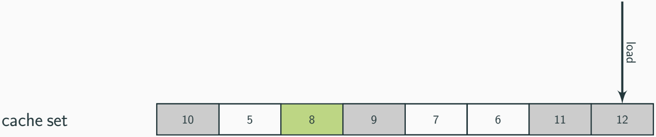


load

cache set	10	5	8	9	7	6	11	4

Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set

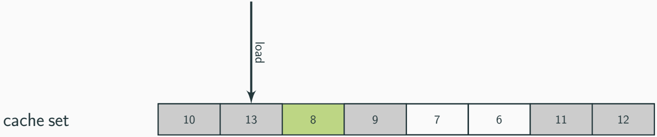


load

cache set	10	5	8	9	7	6	11	12

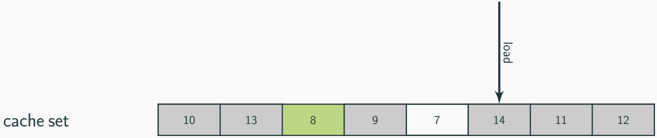
Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set



Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set

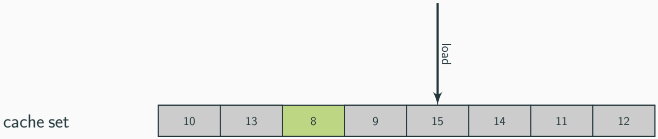


load

cache set	10	13	8	9	7	14	11	12

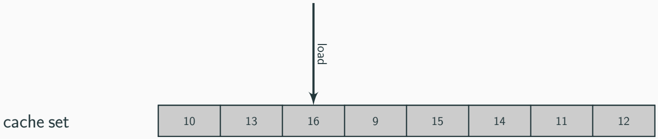
Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set



Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set
- Until the target address is evicted from the cache



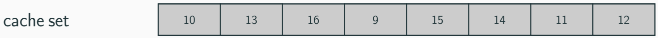
load

cache set

10	13	16	9	15	14	11	12

Cache eviction

- Fill the whole cache → too slow
- Fill a specific cache set
- Until the target address is evicted from the cache



cache set	10	13	16	9	15	14	11	12

→ Ideal case with LRU replacement policy

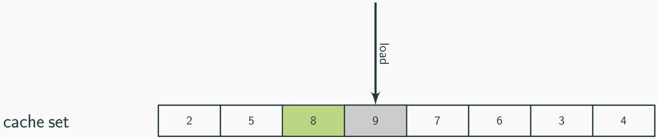
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

cache set	2	5	8	1	7	6	3	4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



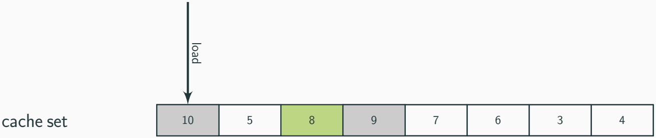
load

cache set

2	5	8	9	7	6	3	4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



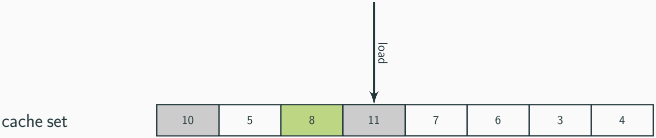
load

cache set

10 | 5 | 8 | 9 | 7 | 6 | 3 | 4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



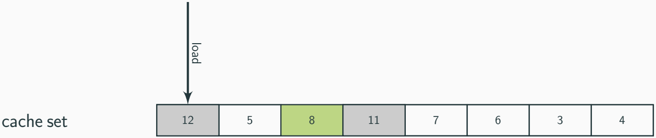
load

cache set

10	5	8	11	7	6	3	4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

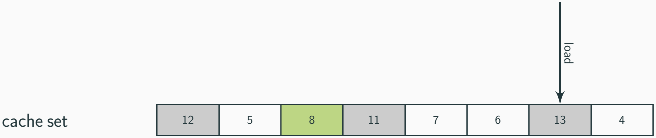


load

cache set	12	5	8	11	7	6	3	4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

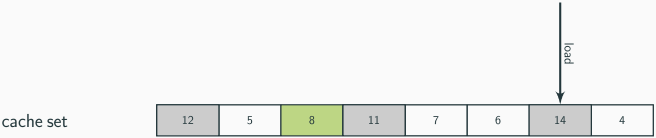


load

cache set								
	12	5	8	11	7	6	13	4

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



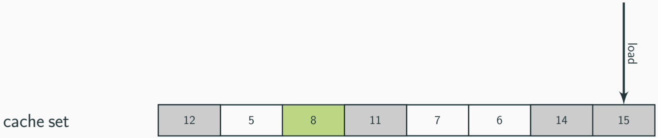
load

cache set

12	5	8	11	7	6	14	4

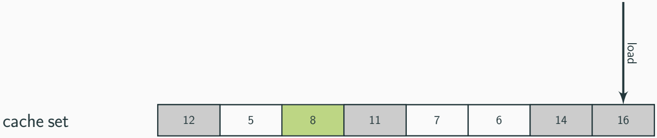
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



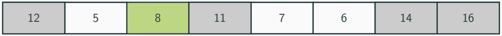
load

cache set

12 5 8 11 7 6 14 16

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



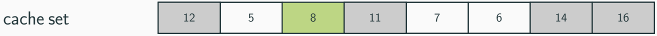
cache set

12	5	8	11	7	6	14	16

→ Simple approach highly inefficient

Cache eviction (what actually happens)

- Pseudo-random cache replacement policy



cache set	12	5	8	11	7	6	14	16

→ Simple approach highly inefficient

Challenge #2

Pseudo-random replacement policy complicates eviction

Timing measurements

- Need fine-grained timing measurements

Timing measurements

- Need fine-grained timing measurements
- Intel x86: Unprivileged `rdtsc` instruction for cycle count

Timing measurements

- Need fine-grained timing measurements
- Intel x86: Unprivileged `rdtsc` instruction for cycle count
- ARM: Cycle counter only in privileged mode

Timing measurements

- Need fine-grained timing measurements
- Intel x86: Unprivileged `rdtsc` instruction for cycle count
- ARM: Cycle counter only in privileged mode

  → Previous attacks required root access

Timing measurements

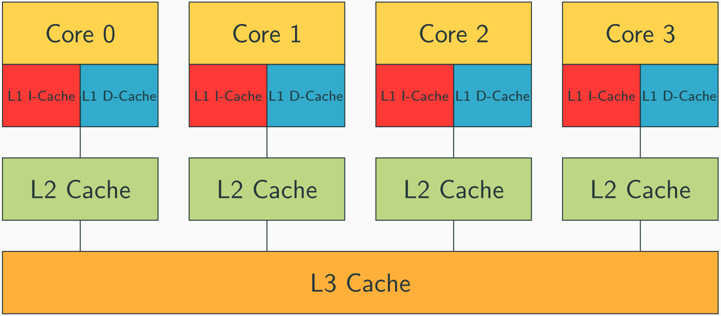
- Need fine-grained timing measurements
- Intel x86: Unprivileged `rdtsc` instruction for cycle count
- ARM: Cycle counter only in privileged mode

  → Previous attacks required root access

Challenge #3

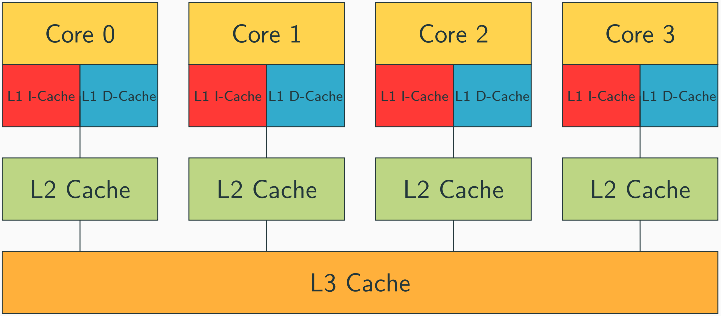
No unprivileged and accurate timing sources

Cache Hierarchy on Intel CPUs



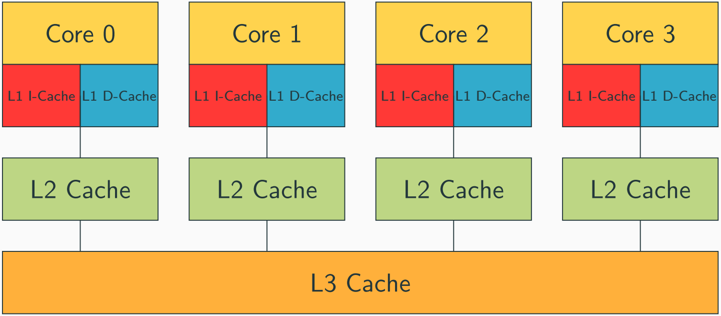
- Last-level cache: L3

Cache Hierarchy on Intel CPUs



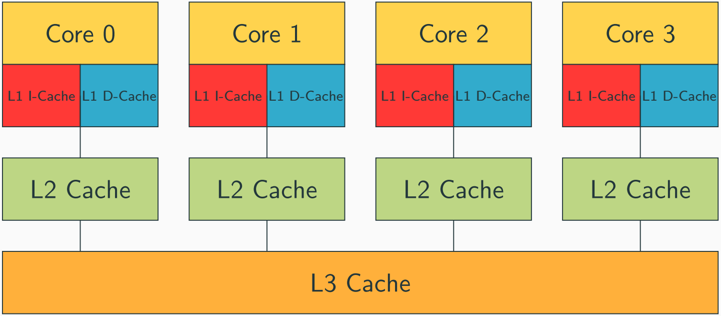
- Last-level cache: L3
    - **shared**

Cache Hierarchy on Intel CPUs



- Last-level cache: L3
    - **shared**
    - **inclusive**

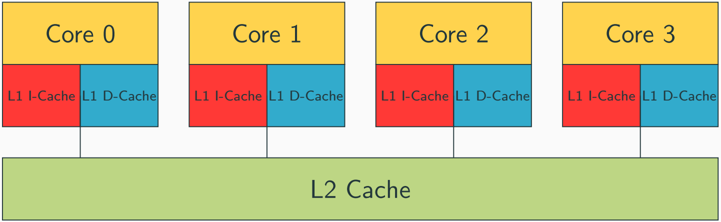
Cache Hierarchy on Intel CPUs



- Last-level cache: L3
  - **shared**
  - **inclusive**

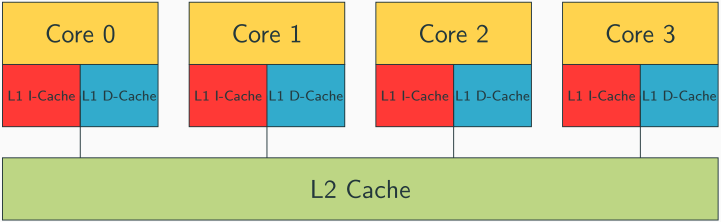
  → Shared memory is shared in the cache across all cores

Cache Hierarchy on ARM Cortex-A CPUs



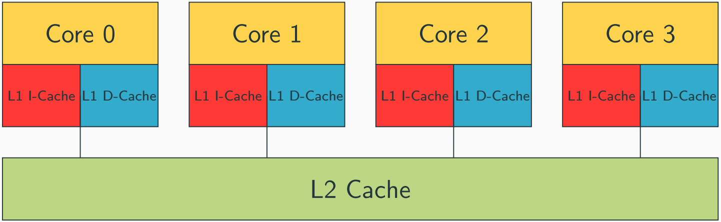
- Last-level cache: L2

Cache Hierarchy on ARM Cortex-A CPUs



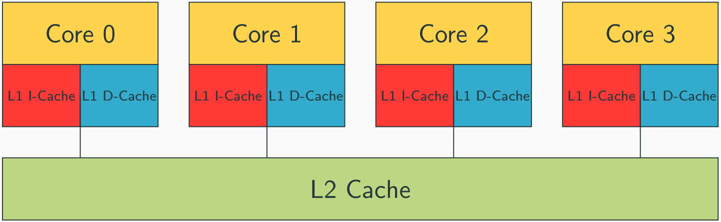
- Last-level cache: L2
  - **shared**

Cache Hierarchy on ARM Cortex-A CPUs



- Last-level cache: L2
  - **shared**
  - not **inclusive**

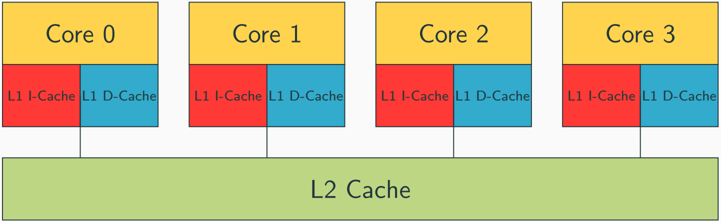
Cache Hierarchy on ARM Cortex-A CPUs



- Last-level cache: L2
    - **shared**
    - not **inclusive**

→ Shared memory that is not in L2 is not shared in the cache.

Cache Hierarchy on ARM Cortex-A CPUs



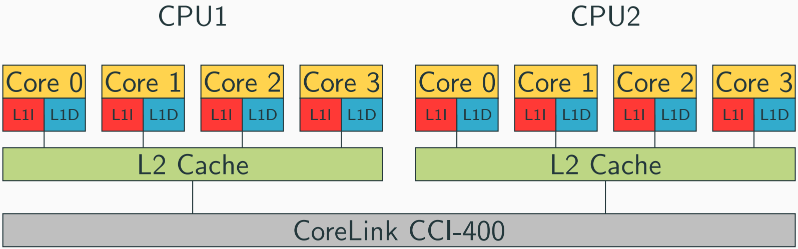
- Last-level cache: L2
  - **shared**
  - not **inclusive**

  → Shared memory that is not in L2 is not shared in the cache.

Challenge #4

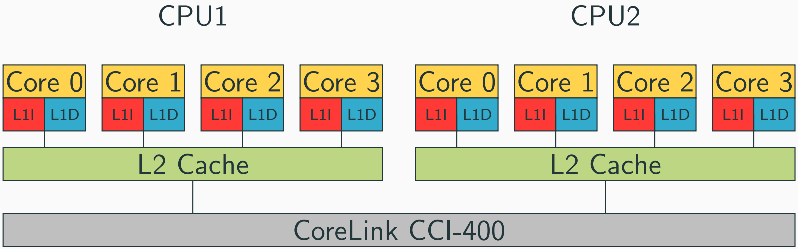
Non-inclusive caches

Cache Hierarchy on ARM big.LITTLE



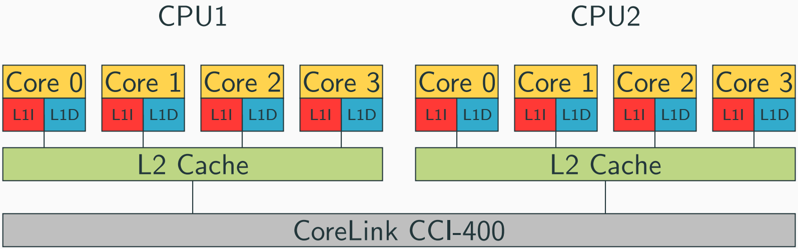
- Interconnects multiple CPUs to combine energy efficiency and performance

Cache Hierarchy on ARM big.LITTLE



- Interconnects multiple CPUs to combine energy efficiency and performance
- CPUs do not share a cache

Cache Hierarchy on ARM big.LITTLE



- Interconnects multiple CPUs to combine energy efficiency and performance
- CPUs do not share a cache

Challenge #5

No shared cache

Let’s solve those challenges

Challenges

Challenge #1 No flush instruction

Challenge #2 Pseudo-random replacement policy

Challenge #3 No unprivileged timing

Challenge #4 Non-inclusive caches

Challenge #5 No shared cache

Solving #1: No flush instruction

- Replace the missing flush instruction with cache eviction

Solving #1: No flush instruction

- Replace the missing flush instruction with cache eviction
- Works on Intel x86

Solving #1: No flush instruction

- Replace the missing flush instruction with cache eviction
- Works on Intel x86
  - *Prime+Probe*
  - *Flush+Reload* → *Evict+Reload*

Challenges

Challenge #1 No flush instruction ✔

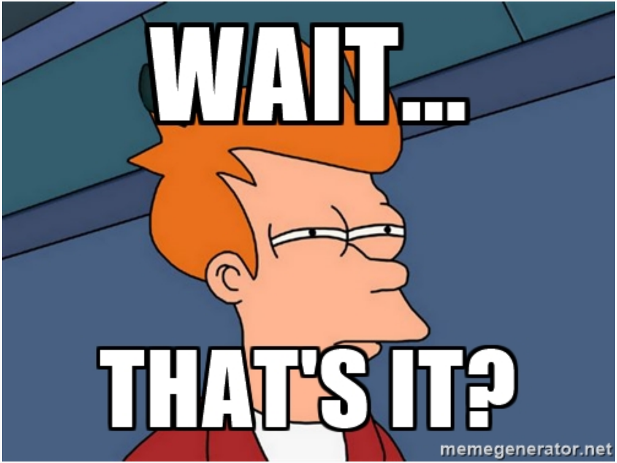
Challenge #2 Pseudo-random replacement policy

Challenge #3 No unprivileged timing

Challenge #4 Non-inclusive caches

Challenge #5 No shared cache

Challenges



Challenges

- No.

Challenges

- No.
- Eviction can be slow and unreliable. . .

Challenges

- No.
- Eviction can be slow and unreliable. . .
- Unless you know how to properly evict data

Challenges

- No.
- Eviction can be slow and unreliable. . .
- Unless you know how to properly evict data

→ Central idea of our Rowhammer.js paper

Solving #2: Pseudo-random replacement policy

- Cache line to be discarded is chosen pseudo-randomly

Solving #2: Pseudo-random replacement policy

- Cache line to be discarded is chosen pseudo-randomly
- Accessing once *n* addresses in an *n*-way cache set

Solving #2: Pseudo-random replacement policy

- Cache line to be discarded is chosen pseudo-randomly
- Accessing once *n* addresses in an *n*-way cache set

  → Cache eviction slow and unreliable

Solving #2: Pseudo-random replacement policy

- Cache line to be discarded is chosen pseudo-randomly
- Accessing once *n* addresses in an *n*-way cache set

  → Cache eviction slow and unreliable

Solution:

- Accessing unique addresses several times, with different access patterns

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗
200	200	33 110 ✗	96.04% ✗

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗
200	200	33 110 ✗	96.04% ✗
800	800	142 876 ✗	99.10% ✓

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗
200	200	33 110 ✗	96.04% ✗
800	800	142 876 ✗	99.10% ✓
21	96	4 275 ✓	99.93% ✓

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗
200	200	33 110 ✗	96.04% ✗
800	800	142 876 ✗	99.10% ✓
21	96	4 275 ✓	99.93% ✓
22	102	5 101 ✓	99.99% ✓

Solving #2: Pseudo-random replacement policy

Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

Addresses	Accesses	Cycles	Eviction rate
48	48	6 517 ✓	70.78% ✗
200	200	33 110 ✗	96.04% ✗
800	800	142 876 ✗	99.10% ✓
21	96	4 275 ✓	99.93% ✓
22	102	5 101 ✓	99.99% ✓
23	190	6 209 ✓	100.0% ✓

Solving #2: Pseudo-random replacement policy

- We fully automated this process

Solving #2: Pseudo-random replacement policy

- We fully automated this process
    - Compile target executable with generated eviction strategy

Solving #2: Pseudo-random replacement policy

- We fully automated this process
  - Compile target executable with generated eviction strategy
  - Execute on target device

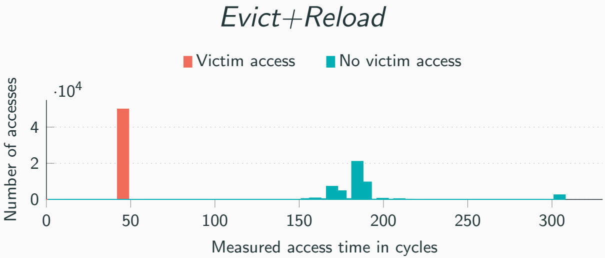
Solving #2: Pseudo-random replacement policy

- We fully automated this process
    - Compile target executable with generated eviction strategy
    - Execute on target device
    - Evaluate log files and build result database

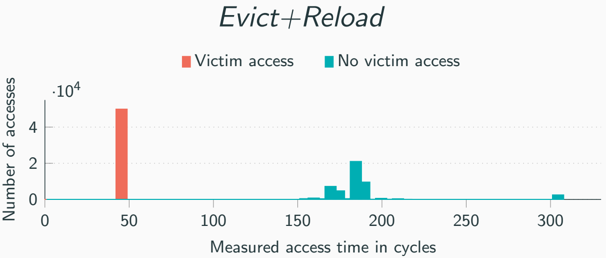
Solving #2: Pseudo-random replacement policy

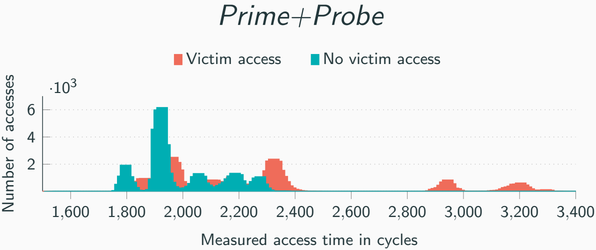
- We fully automated this process
  - Compile target executable with generated eviction strategy
  - Execute on target device
  - Evaluate log files and build result database
- Find fast and efficient eviction strategies for any device

Solving #2: Pseudo-random replacement policy



Solving #2: Pseudo-random replacement policy





Challenges

Challenge #1 No flush instruction ✔

Challenge #2 Pseudo-random replacement policy ✔

Challenge #3 No unprivileged timing

Challenge #4 Non-inclusive caches

Challenge #5 No shared cache

Solving #3: No unprivileged timing

1. Performance counter: Privileged

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
   - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
   - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`
3. `clock_gettime`: Unprivileged `POSIX` function

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
   - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`
3. `clock_gettime`: Unprivileged `POSIX` function
4. Thread counter: Unprivileged, multithreaded

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
    - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`
3. `clock_gettime`: Unprivileged `POSIX` function
4. Thread counter: Unprivileged, multithreaded

- Unprivileged timing sources

Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
   - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`
3. `clock_gettime`: Unprivileged `POSIX` function
4. Thread counter: Unprivileged, multithreaded

- Unprivileged timing sources
- Nanosecond resolution for all sources

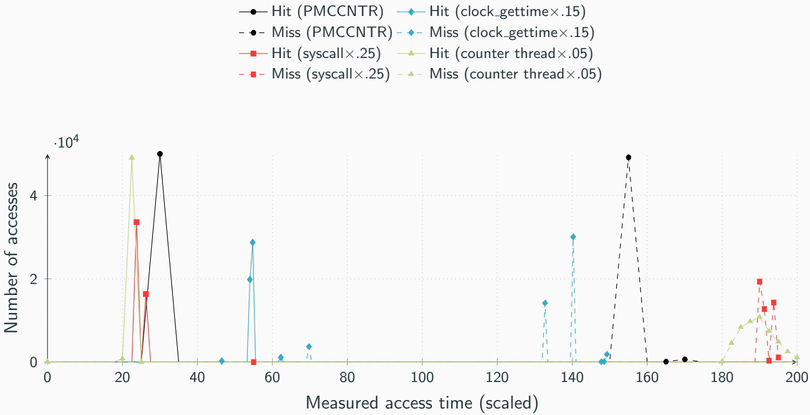
Solving #3: No unprivileged timing

1. Performance counter: Privileged
2. `perf_event_open`: Unprivileged syscall
   - Unavailable on new devices: Privileged or kernel compiled with `CONFIG_PERF=n`
3. `clock_gettime`: Unprivileged `POSIX` function
4. Thread counter: Unprivileged, multithreaded

- Unprivileged timing sources
- Nanosecond resolution for all sources

→ Allows distinguishing cache hits from cache misses

Solving #3: No unprivileged timing



Challenges

Challenge #1 No flush instruction ✔

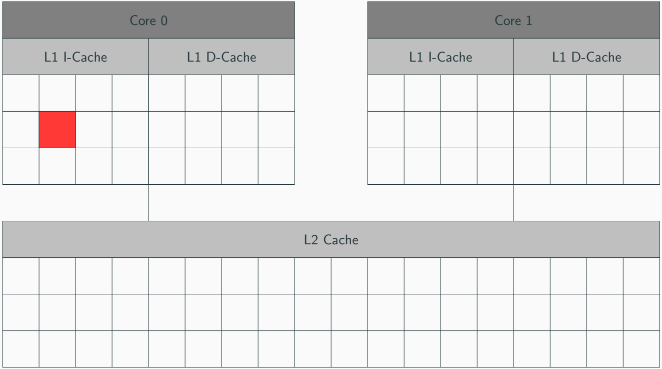
Challenge #2 Pseudo-random replacement policy ✔

Challenge #3 No unprivileged timing ✔

Challenge #4 Non-inclusive caches

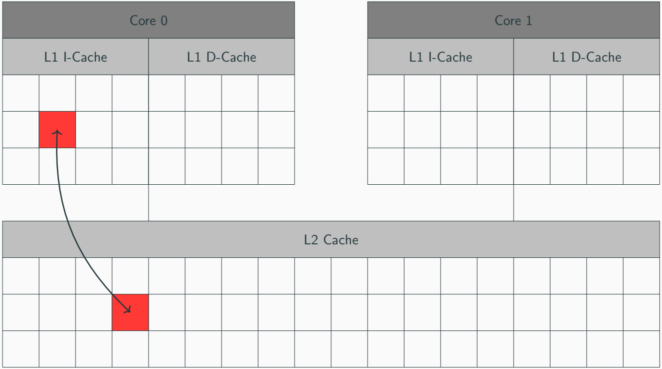
Challenge #5 No shared cache

Solving #4: Non-inclusive caches



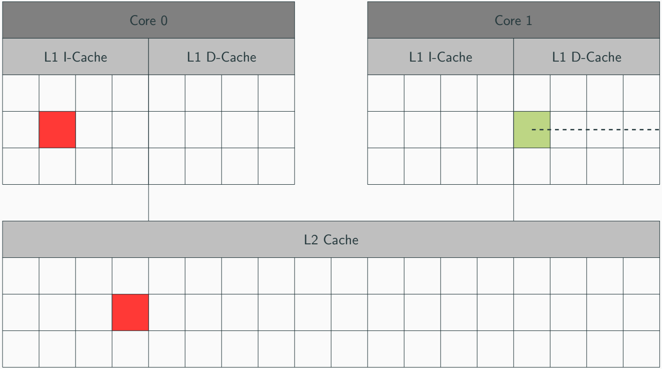
- Instruction-inclusive, data-non-inclusive caches

Solving #4: Non-inclusive caches



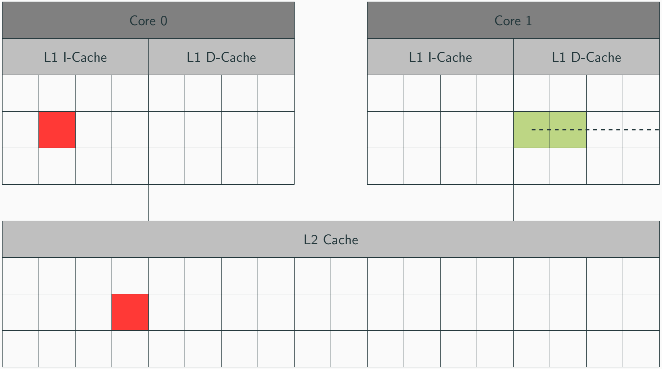
- Instruction-inclusive, data-non-inclusive caches

Solving #4: Non-inclusive caches



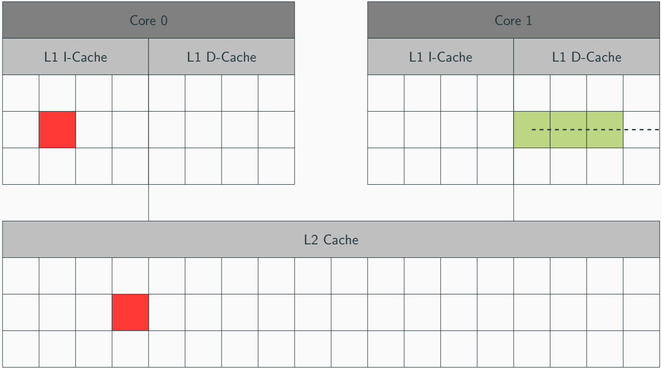
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



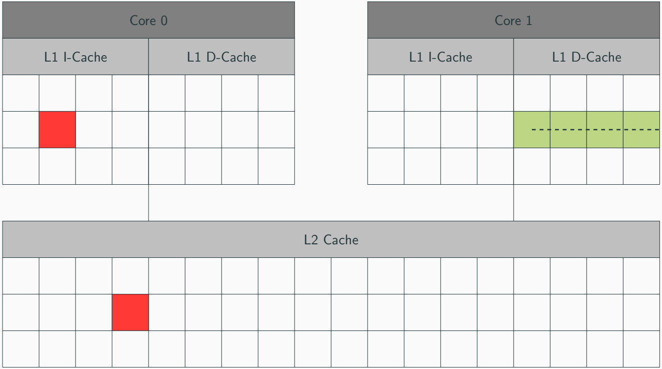
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



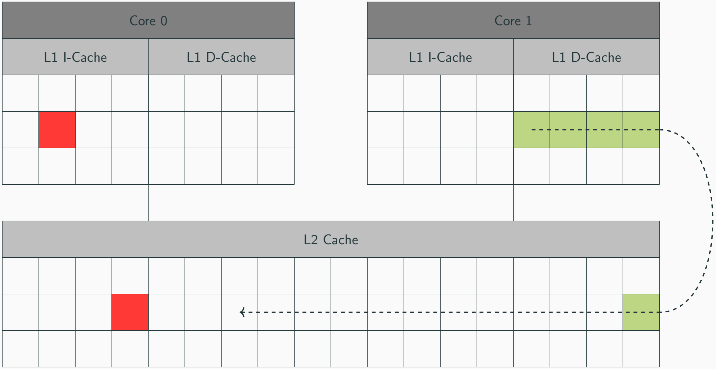
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



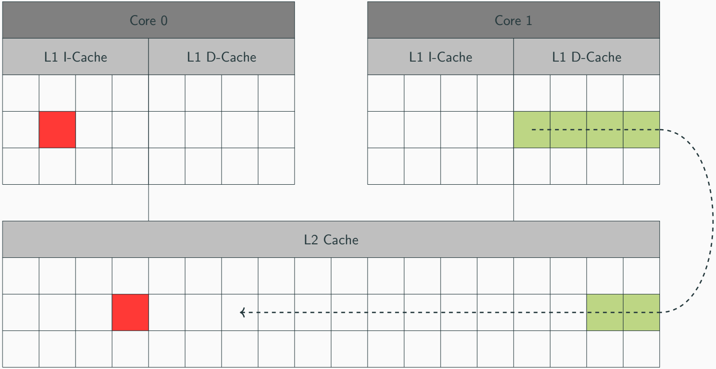
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



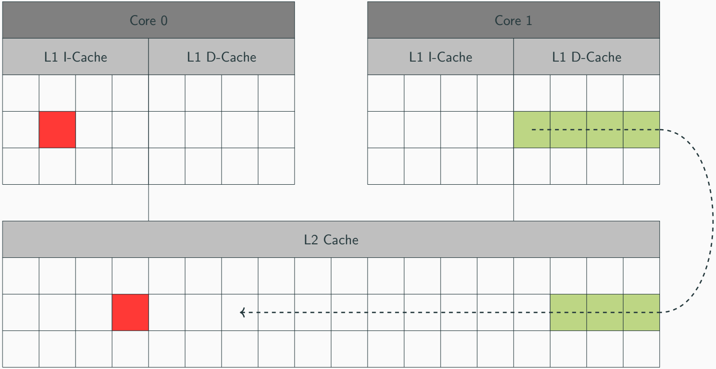
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



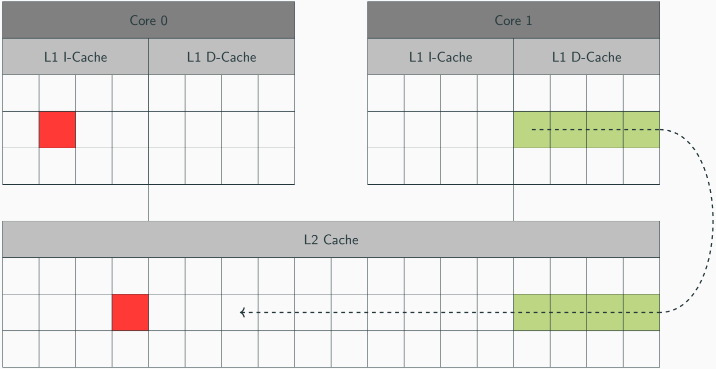
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



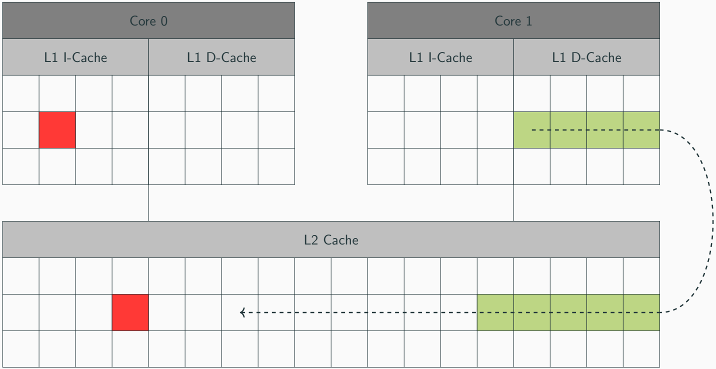
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



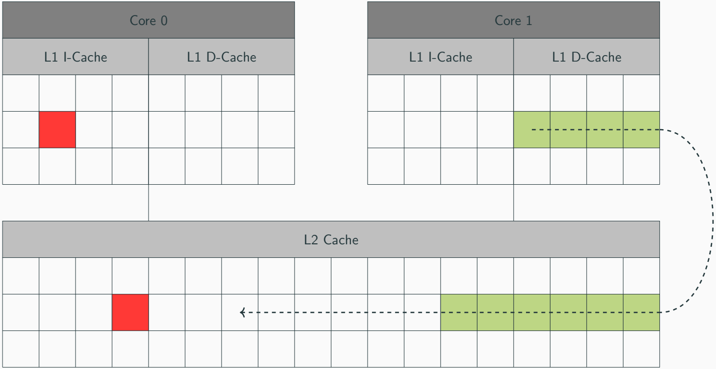
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



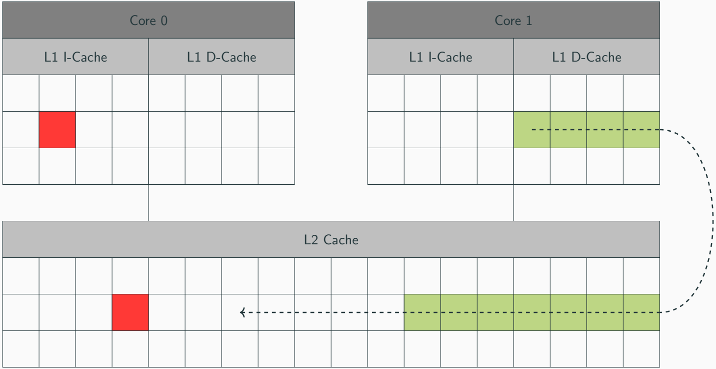
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



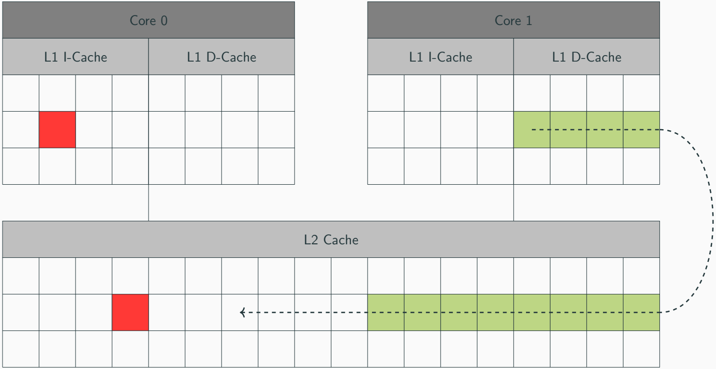
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



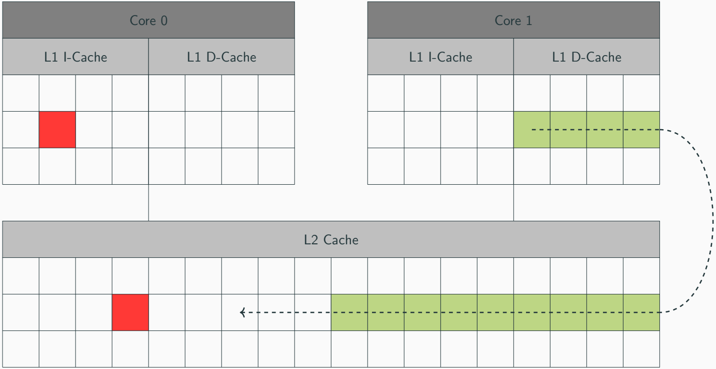
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



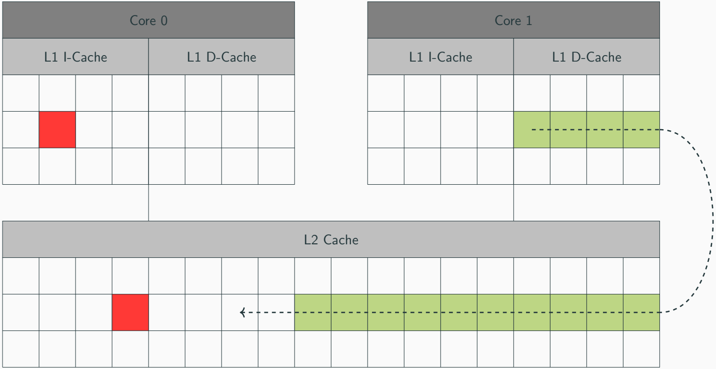
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



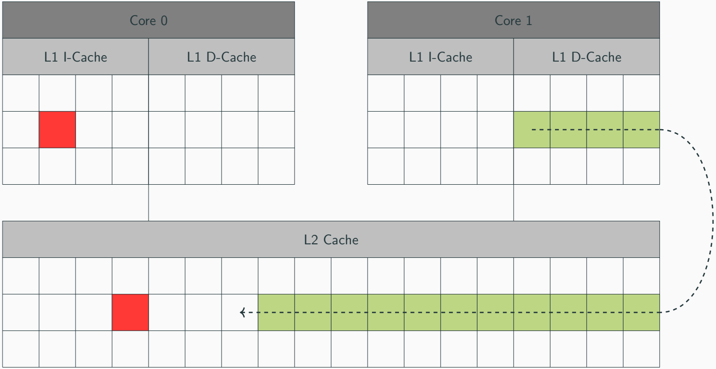
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



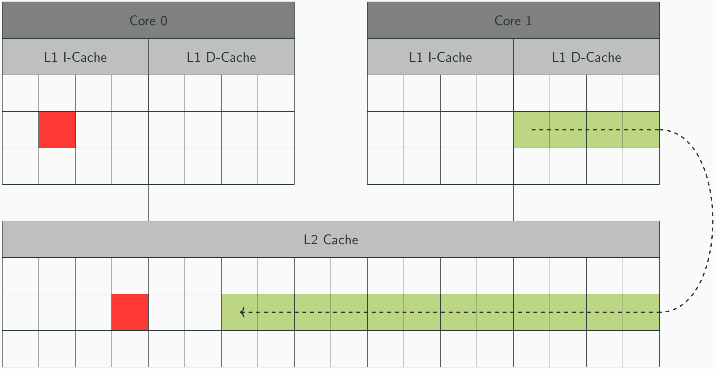
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



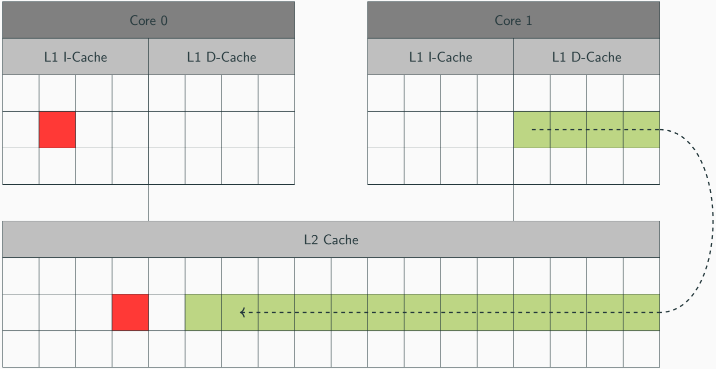
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



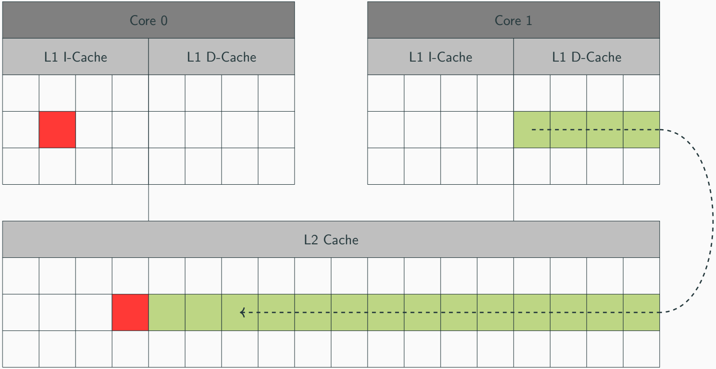
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



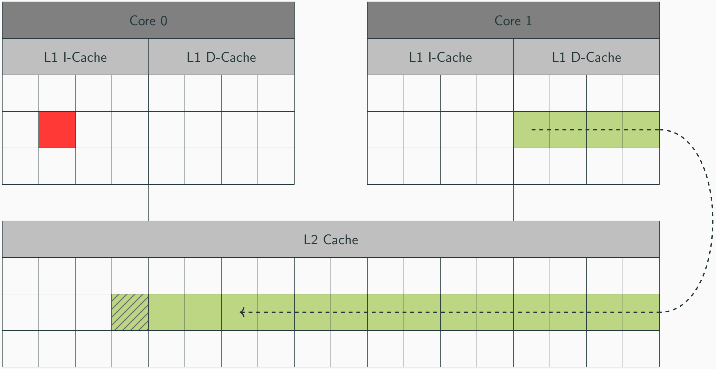
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



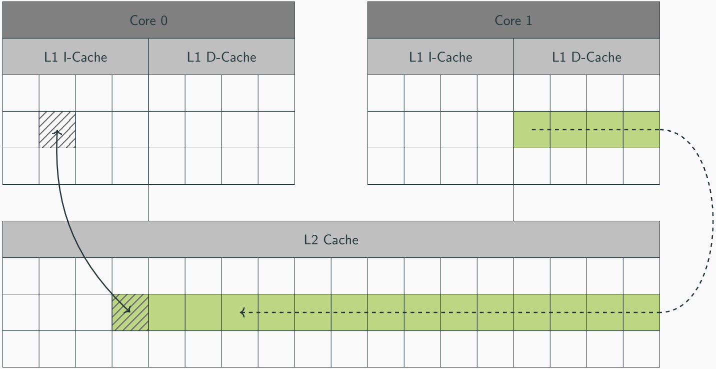
- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache

Solving #4: Non-inclusive caches



- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache
- Inclusion: Line evicted from L2 → also evicted from L1

Solving #4: Non-inclusive caches



- Instruction-inclusive, data-non-inclusive caches
- Fill-up L1 D-Cache and begin to populate shared L2 Cache
- Inclusion: Line evicted from L2 → also evicted from L1

Solving #4: Non-inclusive caches

- How can we determine if the victim has loaded the address?

Solving #4: Non-inclusive caches

- How can we determine if the victim has loaded the address?
- Cache coherency protocol

Solving #4: Non-inclusive caches

- How can we determine if the victim has loaded the address?
- Cache coherency protocol
    - Fetches data from remote cores

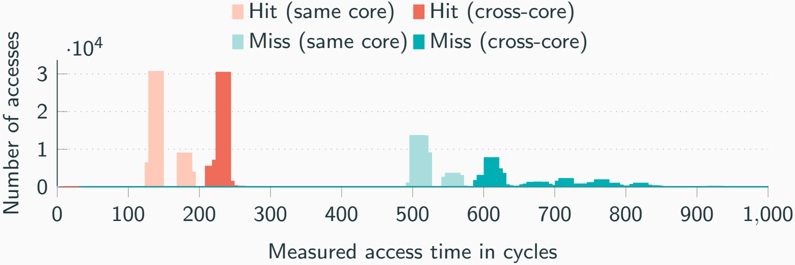
Solving #4: Non-inclusive caches

- How can we determine if the victim has loaded the address?
- Cache coherency protocol
    - Fetches data from remote cores
    - Remote cache hit is faster than DRAM access

Solving #4: Non-inclusive caches

- How can we determine if the victim has loaded the address?
- Cache coherency protocol
    - Fetches data from remote cores
    - Remote cache hit is faster than DRAM access

→ Detect if another core has accessed the memory location



Challenges

Challenge #1 No flush instruction ✔

Challenge #2 Pseudo-random replacement policy ✔

Challenge #3 No unprivileged timing ✔

Challenge #4 Non-inclusive caches ✔

Challenge #5 No shared cache

Solving #5: No shared cache

- Multiple CPUs that do not share a cache

Solving #5: No shared cache

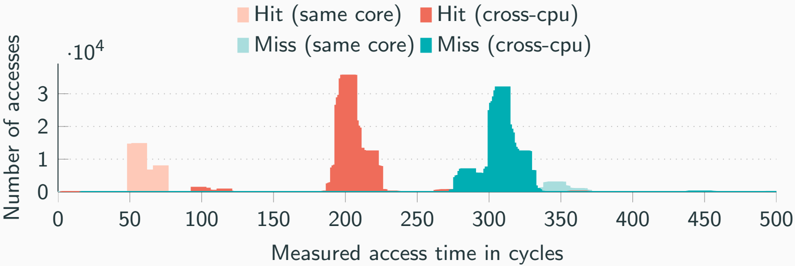
- Multiple CPUs that do not share a cache
- Cache coherency protocol (again)

Solving #5: No shared cache

- Multiple CPUs that do not share a cache
- Cache coherency protocol (again)
    - Fetches data from remote CPU

Solving #5: No shared cache

- Multiple CPUs that do not share a cache
- Cache coherency protocol (again)
  - Fetches data from remote CPU
  - Remote cache hit is faster than DRAM access



Challenges

Challenge #1 No flush instruction ✔

Challenge #2 Pseudo-random replacement policy ✔

Challenge #3 No unprivileged timing ✔

Challenge #4 Non-inclusive caches ✔

Challenge #5 No shared cache ✔

Attack scenarios

Case study #1

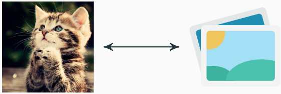
Covert communication

Case Study #1: Covert Channel

- Malicious privacy gallery app

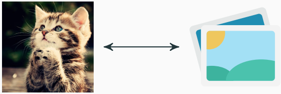
Case Study #1: Covert Channel

- Malicious privacy gallery app
  - No permissions except accessing your images



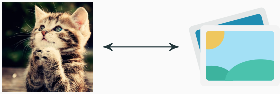
Case Study #1: Covert Channel

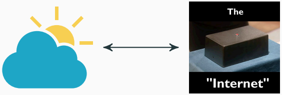
- Malicious privacy gallery app
  - No permissions except accessing your images
- Malicious weather widget



Case Study #1: Covert Channel

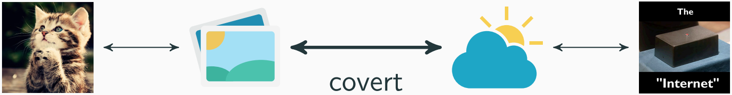
- Malicious privacy gallery app
  - No permissions except accessing your images
- Malicious weather widget
  - No permissions except accessing the Internet





Case Study #1: Covert Channel

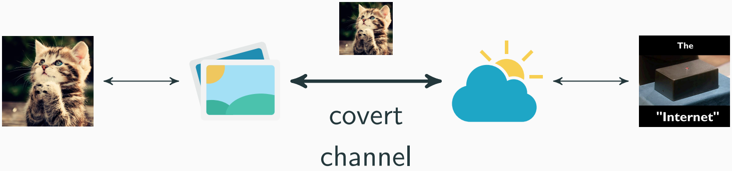
- Malicious privacy gallery app
  - No permissions except accessing your images
- Malicious weather widget
  - No permissions except accessing the Internet



covert channel

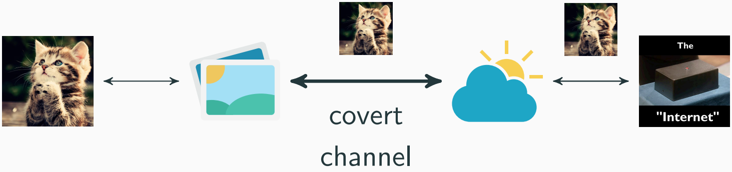
Case Study #1: Covert Channel

- Malicious privacy gallery app
  - No permissions except accessing your images
- Malicious weather widget
  - No permissions except accessing the Internet


covert channel

Case Study #1: Covert Channel

- Malicious privacy gallery app
  - No permissions except accessing your images
- Malicious weather widget
  - No permissions except accessing the Internet



Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
    - No permissions

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
    - No permissions
    - No Intents, Binders, ASHMEM, . . .

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
  - No permissions
  - No Intents, Binders, ASHMEM, . . .
- A covert channel

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
    - No permissions
    - No Intents, Binders, ASHMEM, . . .
- A covert channel
    - Enables two unprivileged apps to communicate

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
  - No permissions
  - No Intents, Binders, ASHMEM, . . .
- A covert channel
  - Enables two unprivileged apps to communicate
  - Does not use data transfer mechanisms provided by the OS

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
  - No permissions
  - No Intents, Binders, ASHMEM, . . .
- A covert channel
  - Enables two unprivileged apps to communicate
  - Does not use data transfer mechanisms provided by the OS
  - Evades the sandboxing concept and permission system

Case Study #1: Covert Channel

- Two apps want to communicate with each other, but are not allowed or not able to do so
    - No permissions
    - No Intents, Binders, ASHMEM, ...
- A covert channel
    - Enables two unprivileged apps to communicate
    - Does not use data transfer mechanisms provided by the OS
    - Evades the sandboxing concept and permission system

→ Collusion attack

Case Study #1: Covert Channel

- Build a covert channel using the cache

Case Study #1: Covert Channel

- Build a covert channel using the cache
    - Using addresses from a shared library/executable

Case Study #1: Covert Channel

- Build a covert channel using the cache
  - Using addresses from a shared library/executable
  - Bits transmitted with cache hits and misses

Case Study #1: Covert Channel

- Build a covert channel using the cache
  - Using addresses from a shared library/executable
  - Bits transmitted with cache hits and misses
- Transmit 0: Do not access the address → cache miss

Case Study #1: Covert Channel

- Build a covert channel using the cache
    - Using addresses from a shared library/executable
    - Bits transmitted with cache hits and misses
- Transmit 0: Do not access the address → cache miss
- Transmit 1: Access the address → cache hit

Case Study #1: Covert Channel

- Build a protocol based on packets

Case Study #1: Covert Channel

- Build a protocol based on packets
    - Use a sequence number (SQN)

Case Study #1: Covert Channel

- Build a protocol based on packets
  - Use a sequence number (SQN)
  - Protect payload and sequence number with a checksum

31 24	23 8	7 0
Sender SQN	Payload	CRC

10 3	2 0
Receiver SQN	CRC

Case Study #1: Covert Channel

- Works using *Flush+Reload*, *Evict+Reload* and *Flush+Flush*

Case Study #1: Covert Channel

- Works using *Flush+Reload*, *Evict+Reload* and *Flush+Flush*
- Works cross-core and cross-CPU

Case Study #1: Covert Channel

- Works using *Flush+Reload*, *Evict+Reload* and *Flush+Flush*
- Works cross-core and cross-CPU
- Faster than state of the art by several orders of magnitude

Case Study #1: Covert Channel

- Works using *Flush+Reload*, *Evict+Reload* and *Flush+Flush*
- Works cross-core and cross-CPU
- Faster than state of the art by several orders of magnitude

Work	Type	Bandwidth [bps]	Error rate
Schlegel et al.	Vibration settings	87	–
Schlegel et al.	Volume settings	150	–
Schlegel et al.	File locks	685	–
Marforio et al.	UNIX socket discovery	2 600	–
Marforio et al.	Type of Intents	4 300	–

Case Study #1: Covert Channel

- Works using *Flush+Reload*, *Evict+Reload* and *Flush+Flush*
- Works cross-core and cross-CPU
- Faster than state of the art by several orders of magnitude

Work	Type	Bandwidth [bps]	Error rate
Schlegel et al.	Vibration settings	87	–
Schlegel et al.	Volume settings	150	–
Schlegel et al.	File locks	685	–
Marforio et al.	UNIX socket discovery	2 600	–
Marforio et al.	Type of Intents	4 300	–
Ours (OnePlus One)	*Evict+Reload*, cross-core	12 537	5.00%
Ours (Alcatel One Touch Pop 2)	*Evict+Reload*, cross-core	13 618	3.79%
Ours (Samsung Galaxy S6)	*Flush+Flush*, cross-core	178 292	0.48%
Ours (Samsung Galaxy S6)	*Flush+Reload*, cross-CPU	257 509	1.83%
Ours (Samsung Galaxy S6)	*Flush+Reload*, cross-core	1 140 650	1.10%

Case study #2

Spying on the user

Case Study #2: Spying on the User

- Issue: Locating event-dependent memory access

Case Study #2: Spying on the User

- Issue: Locating event-dependent memory access

→ Cache Template Attacks

Case Study #2: Spying on the User

- Issue: Locating event-dependent memory access

→ Cache Template Attacks

1. Shared library or executable is mapped

Case Study #2: Spying on the User

- Issue: Locating event-dependent memory access

→ Cache Template Attacks

1. Shared library or executable is mapped
2. Trigger an event in parallel and *Flush+Reload* one address

Case Study #2: Spying on the User

- Issue: Locating event-dependent memory access

→ Cache Template Attacks

1. Shared library or executable is mapped
2. Trigger an event in parallel and *Flush+Reload* one address
   → Cache hit: Address used by the library/executable

Case Study #2: Spying on the User

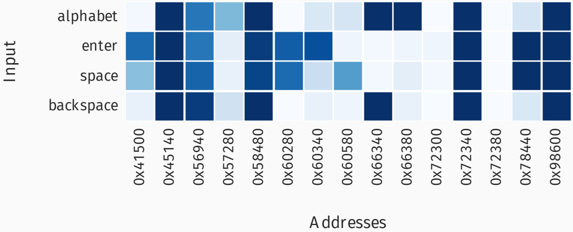
- Issue: Locating event-dependent memory access

→ Cache Template Attacks

1. Shared library or executable is mapped
2. Trigger an event in parallel and *Flush+Reload* one address
   → Cache hit: Address used by the library/executable
3. Repeat step 2 for every address

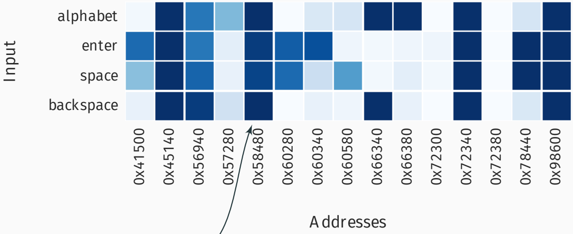
Case Study #2: Spying on the User

- Cache template matrix
  = How many cache hits for each pair (event, address)?
- On shared library and ART binaries, e.g., AOSP keyboard



Case Study #2: Spying on the User

- Cache template matrix
  = How many cache hits for each pair (event, address)?
- On shared library and ART binaries, e.g., AOSP keyboard



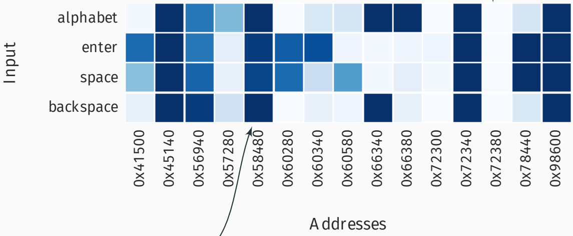
Many cache hits

Case Study #2: Spying on the User

- Cache template matrix

  = How many cache hits for each pair (event, address)?
- On shared library and ART binaries, e.g., AOSP keyboard

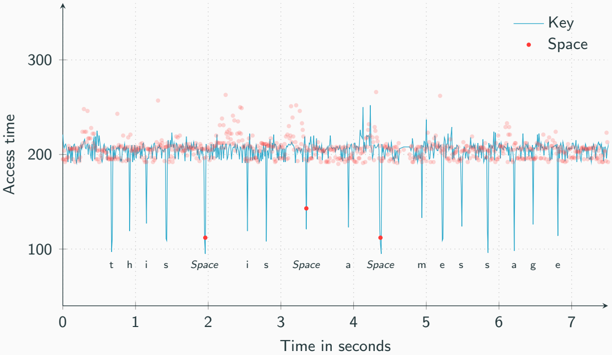
No cache hits



Many cache hits

Case Study #2: Spying on the User

Evict+Reload on two addresses on the Alcatel One Touch Pop 2 in `custpack@app@withoutlibs@LatinIME.apk@classes.dex`
→ Differentiate keys from spaces



Case Study #2: Spying on the User

- Endless possibilities

Case Study #2: Spying on the User

- Endless possibilities
- Scan all the libraries

Case Study #2: Spying on the User

- Endless possibilities
- Scan all the libraries
  - Find all secret-dependent accesses and automate attacks

Case Study #2: Spying on the User

- Endless possibilities
- Scan all the libraries
    - Find all secret-dependent accesses and automate attacks
    - No need for source code

Case Study #2: Spying on the User

- Endless possibilities
- Scan all the libraries
  - Find all secret-dependent accesses and automate attacks
  - No need for source code
- Spy and learn about user's behavior

Case study #3

Attacking cryptographic algorithms

Case Study #3: Attacking Cryptographic Algorithms

- AES T-Tables: Fast software implementation

Case Study #3: Attacking Cryptographic Algorithms

- AES T-Tables: Fast software implementation
- Uses precomputed look-up tables

Case Study #3: Attacking Cryptographic Algorithms

- AES T-Tables: Fast software implementation
- Uses precomputed look-up tables
- One-round known-plaintext attack by Osvik et al. (2006)
  - p plaintext and k secret key
  - Intermediate state $x^{(r)} = (x_0^{(r)}, \ldots, x_{15}^{(r)})$ at each round r
  - First round, accessed table indices are

$$x_i^{(0)} = p_i \oplus k_i \qquad \text{for all } i = 0, \ldots, 15$$

Case Study #3: Attacking Cryptographic Algorithms

- AES T-Tables: Fast software implementation
- Uses precomputed look-up tables
- One-round known-plaintext attack by Osvik et al. (2006)
  - p plaintext and k secret key
  - Intermediate state $x^{(r)} = (x_0^{(r)}, \ldots, x_{15}^{(r)})$ at each round r
  - First round, accessed table indices are

$$x_i^{(0)} = p_i \oplus k_i \qquad \text{for all } i = 0, \ldots, 15$$

→ Recovering accessed table indices ⇒ recovering the key

Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables

Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables

→ Let's monitor which T-Table entry is accessed!

Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables

→ Let's monitor which T-Table entry is accessed!

- Java VM creates a copy of the T-tables when the app starts

Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables

→ Let's monitor which T-Table entry is accessed!

- Java VM creates a copy of the T-tables when the app starts
- No shared memory → no *Evict+Reload* or *Flush+Reload*

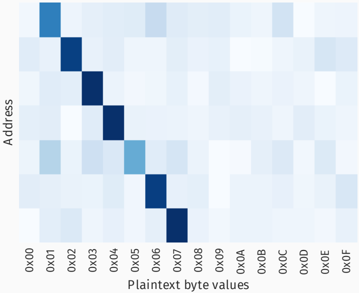
Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables

→ Let's monitor which T-Table entry is accessed!

- Java VM creates a copy of the T-tables when the app starts
- No shared memory → no *Evict+Reload* or *Flush+Reload*

→ *Prime+Probe* to the rescue!



Case study #4

Monitoring ARM TrustZone

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment
    - Roots of Trust

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment
    - Roots of Trust
- Applications (trustlets) running in a secure world

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
  - Secure Execution Environment
  - Roots of Trust
- Applications (trustlets) running in a secure world
  - Credential-store

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment
    - Roots of Trust
- Applications (trustlets) running in a secure world
    - Credential-store
    - Secure element for payments

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment
    - Roots of Trust
- Applications (trustlets) running in a secure world
    - Credential-store
    - Secure element for payments
    - Digital Rights Management (DRM)

Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology
    - Secure Execution Environment
    - Roots of Trust
- Applications (trustlets) running in a secure world
    - Credential-store
    - Secure element for payments
    - Digital Rights Management (DRM)
- Information from the trusted world should not be leaked to the non-secure world

Case Study #4: Monitoring ARM TrustZone

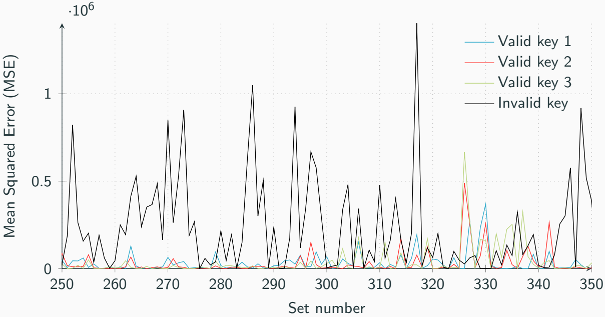
- Timing difference for different RSA signature keys

Case Study #4: Monitoring ARM TrustZone

- Timing difference for different RSA signature keys
- No knowledge of the TrustZone/trustlet implementation

Case Study #4: Monitoring ARM TrustZone

- Timing difference for different RSA signature keys
- No knowledge of the TrustZone/trustlet implementation
- Valid keys and invalid keys are distinguishable



Tools

libflush

- Library to build cross-platform cache attacks

libflush

- Library to build cross-platform cache attacks
  - x86
  - ARMv7
  - ARMv8

libflush

- Library to build cross-platform cache attacks
  - x86
  - ARMv7
  - ARMv8
- Implements various attack techniques
  - *Flush+Reload*
  - *Evict+Reload*
  - *Prime+Probe*
  - *Flush+Flush*
  - *Prefetch*

libflush

- Library to build cross-platform cache attacks
    - x86
    - ARMv7
    - ARMv8
- Implements various attack techniques
    - *Flush+Reload*
    - *Evict+Reload*
    - *Prime+Probe*
    - *Flush+Flush*
    - *Prefetch*
- Open Source

libflush

- All described examples have been developed on top of `libflush`

libflush

- All described examples have been developed on top of `libflush`
- Includes eviction strategies for several devices

libflush

- All described examples have been developed on top of `libflush`
- Includes eviction strategies for several devices
- Comes with example code and documentation

libflush

- All described examples have been developed on top of `libflush`
- Includes eviction strategies for several devices
- Comes with example code and documentation
- Even allows to implement cross-platform Rowhammer attacks

libflush

- All described examples have been developed on top of `libflush`
- Includes eviction strategies for several devices
- Comes with example code and documentation
- Even allows to implement cross-platform Rowhammer attacks

`github.com/iaik/armageddon/libflush`

Eviction Strategy Evaluator

- Utilizes `libflush` to evaluate eviction strategies

Eviction Strategy Evaluator

- Utilizes `libflush` to evaluate eviction strategies
- Automatically executed on target platform

Eviction Strategy Evaluator

- Utilizes `libflush` to evaluate eviction strategies
- Automatically executed on target platform
- Results are stored in a database file

Eviction Strategy Evaluator

- Utilizes `libflush` to evaluate eviction strategies
- Automatically executed on target platform
- Results are stored in a database file

github.com/iaik/armageddon/eviction_strategy_evaluator

Cache Template Attacks

- Cross-platform cache template attack tool

Cache Template Attacks

- Cross-platform cache template attack tool
- Scan libraries and executable for vulnerable addresses

Cache Template Attacks

- Cross-platform cache template attack tool
- Scan libraries and executable for vulnerable addresses
- Additional tool to simulate input events

Cache Template Attacks

- Cross-platform cache template attack tool
- Scan libraries and executable for vulnerable addresses
- Additional tool to simulate input events

`github.com/iaik/armageddon/cache_template_attacks`

Countermeasures

Countermeasures

- Coarse-grained timers

Countermeasures

- Coarse-grained timers
  - → But a thread timer is sufficient…

Countermeasures

- Coarse-grained timers
  → But a thread timer is sufficient…
- No shared memory

Countermeasures

- Coarse-grained timers
  - → But a thread timer is sufficient…
- No shared memory
  - → What about Prime+Probe?

Countermeasures

- Coarse-grained timers
  → But a thread timer is sufficient…
- No shared memory
  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`

Countermeasures

- Coarse-grained timers

  → But a thread timer is sufficient...
- No shared memory

  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`

  → Harder but still possible

Countermeasures

- Coarse-grained timers

  → But a thread timer is sufficient…
- No shared memory

  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`

  → Harder but still possible
- Use cryptographic instruction extensions

Countermeasures

- Coarse-grained timers
  → But a thread timer is sufficient. . .
- No shared memory
  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`
  → Harder but still possible
- Use cryptographic instruction extensions
  → Still not the default everywhere. . .

Countermeasures

- Coarse-grained timers
  - → But a thread timer is sufficient...
- No shared memory
  - → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`
  - → Harder but still possible
- Use cryptographic instruction extensions
  - → Still not the default everywhere...
  - → Doesn't protect from spying user behavior...

Countermeasures

- Coarse-grained timers
  → But a thread timer is sufficient...
- No shared memory
  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`
  → Harder but still possible
- Use cryptographic instruction extensions
  → Still not the default everywhere...
  → Doesn't protect from spying user behavior...

→ Protect crypto.

Countermeasures

- Coarse-grained timers
  → But a thread timer is sufficient…
- No shared memory
  → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`
  → Harder but still possible
- Use cryptographic instruction extensions
  → Still not the default everywhere…
  → Doesn't protect from spying user behavior…

→ Protect crypto. For the rest…

Countermeasures

- Coarse-grained timers
  - → But a thread timer is sufficient...
- No shared memory
  - → What about Prime+Probe?
- Restrict system information, e.g., `pagemap`
  - → Harder but still possible
- Use cryptographic instruction extensions
  - → Still not the default everywhere...
  - → Doesn't protect from spying user behavior...

→ Protect crypto. For the rest... No satisfying solution

Conclusion

Conclusion

- All powerful cache attacks are applicable to mobile devices
  - Without any permissions or privileges

Conclusion

- All powerful cache attacks are applicable to mobile devices
    - Without any permissions or privileges
- Building fast covert channels

Conclusion

- All powerful cache attacks are applicable to mobile devices
    - Without any permissions or privileges
- Building fast covert channels
- Spying on user activity with a high accuracy

Conclusion

- All powerful cache attacks are applicable to mobile devices
  - Without any permissions or privileges
- Building fast covert channels
- Spying on user activity with a high accuracy
- Deriving cryptographic keys

Conclusion

- All powerful cache attacks are applicable to mobile devices
    - Without any permissions or privileges
- Building fast covert channels
- Spying on user activity with a high accuracy
- Deriving cryptographic keys
- ARM TrustZone leaking through the cache

Conclusion

- All powerful cache attacks are applicable to mobile devices
    - Without any permissions or privileges
- Building fast covert channels
- Spying on user activity with a high accuracy
- Deriving cryptographic keys
- ARM TrustZone leaking through the cache
- Try our tools yourself!

ARMageddon

How Your Smartphone CPU Breaks Software-Level Security And Privacy

Moritz Lipp and Clémentine Maurice

November 3, 2016—Black Hat Europe