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
- [twitter](https://twitter.com/mlqxyz)
- [email](mailto: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 research team

- Daniel Gruss
- Raphael Spreitzer
- Stefan Mangard

From Graz University of Technology
Demo
Outline

- Background information
• 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
• Background information
• What are the challenges for cache attacks on ARM?
• How to solve those challenges
• Attack scenarios
• Tools
Cache Attacks
- Data can reside in
Memory Hierarchy

- Data can reside in
  - CPU registers
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
• 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

<table>
<thead>
<tr>
<th>Address</th>
<th>0</th>
<th>16 17</th>
<th>25 26</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Victim address space

Cache

Attacker address space
**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
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
  - 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
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 $\rightarrow$ too slow
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

2 5 8 9 7 6 3 4
Cache eviction

- Fill the whole cache $\rightarrow$ too slow
- Fill a specific cache set
Cache eviction

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

```
cache set
```

```
10 5 8 9 7 6 11 4
load
```
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
Cache eviction

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

![Diagram showing cache eviction process]
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
Cache eviction

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

$\rightarrow$ 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

<table>
<thead>
<tr>
<th>cache set</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

![Cache set diagram]

Simple approach highly inefficient

Challenge #2

Pseudo-random replacement policy complicates eviction
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

Simple approach highly inefficient

Challenge #2

Pseudo-random replacement policy complicates eviction

```
+---+---+---+---+---+---+---+
| 10|  5|  8| 11|  7|  6|  3|  4 |
   |   |   |   |   |   |   |   |
```
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

```
12 5 8 11 7 6 13 4
```

Simple approach highly inefficient
Challenge #2
Pseudo-random replacement policy complicates eviction
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

![Cache Set Diagram]

Simple approach highly inefficient

Challenge #2
Pseudo-random replacement policy complicates eviction
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

![Cache Set Diagram]
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

→ Simple approach highly inefficient
Cache eviction (what actually happens)

- Pseudo-random cache replacement policy

→ 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

Previous attacks required root access

Challenge #3

No unprivileged and accurate timing sources
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

- Core 0
  - L1 I-Cache
  - L1 D-Cache
  - L2 Cache
- Core 1
  - L1 I-Cache
  - L1 D-Cache
  - L2 Cache
- Core 2
  - L1 I-Cache
  - L1 D-Cache
  - L2 Cache
- Core 3
  - L1 I-Cache
  - L1 D-Cache
  - L2 Cache

L3 Cache

- Last-level cache: L3
• 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
• Last-level cache: L2
  • shared
Cache Hierarchy on ARM Cortex-A CPUs

- Last-level cache: L2
  - shared
  - not inclusive
- 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
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 $\rightarrow$ 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

WAIT...

THAT'S IT?

memegenerator.net
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
  \[ \rightarrow \] 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
  $\rightarrow$ 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

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6517</td>
<td>✔️ 70.78%</td>
</tr>
</tbody>
</table>


Solving #2: Pseudo-random replacement policy

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

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6 517</td>
<td>✓ 70.78% X</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>33 110</td>
<td>X 96.04% X</td>
</tr>
</tbody>
</table>
### Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6517</td>
<td>70.78%</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>33110</td>
<td>96.04%</td>
</tr>
<tr>
<td>800</td>
<td>800</td>
<td>142876</td>
<td>99.10%</td>
</tr>
</tbody>
</table>
**Solving #2: Pseudo-random replacement policy**

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

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6,517</td>
<td>✓ 70.78%</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>33,110</td>
<td>✗ 96.04%</td>
</tr>
<tr>
<td>800</td>
<td>800</td>
<td>142,876</td>
<td>✗ 99.10%</td>
</tr>
<tr>
<td>21</td>
<td>96</td>
<td>4,275</td>
<td>✓ 99.93%</td>
</tr>
</tbody>
</table>
Table 1: Different eviction strategies for the Alcatel One Touch Pop 2

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6 517 ✓</td>
<td>70.78% ✗</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>33 110 ✗</td>
<td>96.04% ✗</td>
</tr>
<tr>
<td>800</td>
<td>800</td>
<td>142 876 ✗</td>
<td>99.10% ✓</td>
</tr>
<tr>
<td>21</td>
<td>96</td>
<td>4 275 ✓</td>
<td>99.93% ✓</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>5 101 ✓</td>
<td>99.99% ✓</td>
</tr>
</tbody>
</table>
**Table 1:** Different eviction strategies for the Alcatel One Touch Pop 2

<table>
<thead>
<tr>
<th>Addresses</th>
<th>Accesses</th>
<th>Cycles</th>
<th>Eviction rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>48</td>
<td>6 517</td>
<td>✓ 70.78%</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>33 110</td>
<td>✕ 96.04%</td>
</tr>
<tr>
<td>800</td>
<td>800</td>
<td>142 876</td>
<td>✕ 99.10%</td>
</tr>
<tr>
<td>21</td>
<td>96</td>
<td>4 275</td>
<td>✓ 99.93%</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>5 101</td>
<td>✓ 99.99%</td>
</tr>
<tr>
<td>23</td>
<td>190</td>
<td>6 209</td>
<td>✓ 100.0%</td>
</tr>
</tbody>
</table>
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

**Evict+Reload**

Measured access time in cycles

- **Victim access**
- **No victim access**

Number of accesses
Solving #2: Pseudo-random replacement policy

**Evict+Reload**

Measured access time in cycles

Number of accesses

Victim access

No victim access

**Prime+Probe**

Measured access time in cycles

Number of accesses

Victim access

No victim 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
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
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

![Graph showing measured access time and number of accesses for different operations.]
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
• 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
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 $\rightarrow$ 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

![Diagram showing measured access time in cycles with different categories: Hit (same core), Hit (cross-cpu), Miss (same core), Miss (cross-cpu).]
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
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
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

Transmit 0: Do not access the address → cache miss
Transmit 1: Access the address → cache hit
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 $\rightarrow$ 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

![Protocol Diagram]

<table>
<thead>
<tr>
<th>31</th>
<th>24 23</th>
<th>8 7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sender SQN</td>
<td>Payload</td>
<td>CRC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>10</th>
<th>3 2 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiver SQN</td>
<td>CRC</td>
</tr>
</tbody>
</table>
Case Study #1: Covert Channel

- Works using \textit{Flush+Reload}, \textit{Evict+Reload} and \textit{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 \textit{Flush+Reload}, \textit{Evict+Reload} and \textit{Flush+Flush}
- Works cross-core and cross-CPU
- \textbf{Faster} than state of the art by several orders of magnitude

<table>
<thead>
<tr>
<th>Work</th>
<th>Type</th>
<th>Bandwidth [bps]</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schlegel et al.</td>
<td>Vibration settings</td>
<td>87</td>
<td>–</td>
</tr>
<tr>
<td>Schlegel et al.</td>
<td>Volume settings</td>
<td>150</td>
<td>–</td>
</tr>
<tr>
<td>Schlegel et al.</td>
<td>File locks</td>
<td>685</td>
<td>–</td>
</tr>
<tr>
<td>Marforio et al.</td>
<td>UNIX socket discovery</td>
<td>2 600</td>
<td>–</td>
</tr>
<tr>
<td>Marforio et al.</td>
<td>Type of Intents</td>
<td>4 300</td>
<td>–</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Work</th>
<th>Type</th>
<th>Bandwidth [bps]</th>
<th>Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schlegel et al.</td>
<td>Vibration settings</td>
<td>87</td>
<td>–</td>
</tr>
<tr>
<td>Schlegel et al.</td>
<td>Volume settings</td>
<td>150</td>
<td>–</td>
</tr>
<tr>
<td>Schlegel et al.</td>
<td>File locks</td>
<td>685</td>
<td>–</td>
</tr>
<tr>
<td>Marforio et al.</td>
<td>UNIX socket discovery</td>
<td>2600</td>
<td>–</td>
</tr>
<tr>
<td>Marforio et al.</td>
<td>Type of Intents</td>
<td>4300</td>
<td>–</td>
</tr>
<tr>
<td>Ours (OnePlus One)</td>
<td><em>Evict+Reload</em>, cross-core</td>
<td>12 537</td>
<td>5.00%</td>
</tr>
<tr>
<td>Ours (Alcatel One Touch Pop 2)</td>
<td><em>Evict+Reload</em>, cross-core</td>
<td>13 618</td>
<td>3.79%</td>
</tr>
<tr>
<td>Ours (Samsung Galaxy S6)</td>
<td><em>Flush+Flush</em>, cross-core</td>
<td>178 292</td>
<td>0.48%</td>
</tr>
<tr>
<td>Ours (Samsung Galaxy S6)</td>
<td><em>Flush+Reload</em>, cross-CPU</td>
<td>257 509</td>
<td>1.83%</td>
</tr>
<tr>
<td>Ours (Samsung Galaxy S6)</td>
<td><em>Flush+Reload</em>, cross-core</td>
<td>1 140 650</td>
<td>1.10%</td>
</tr>
</tbody>
</table>
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
  \rightarrow 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 \textit{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 \textit{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

<table>
<thead>
<tr>
<th>Addresses</th>
<th>0x41500</th>
<th>0x45140</th>
<th>0x456940</th>
<th>0x57280</th>
<th>0x58480</th>
<th>0x60280</th>
<th>0x60340</th>
<th>0x60580</th>
<th>0x66340</th>
<th>0x66380</th>
<th>0x72300</th>
<th>0x72340</th>
<th>0x72380</th>
<th>0x78440</th>
<th>0x98600</th>
</tr>
</thead>
<tbody>
<tr>
<td>alphabet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enter</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>space</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>backspace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

Many cache hits

No 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

  - p plaintext and k secret key
  - Intermediate state \( x(r) = (x(r)_0, \ldots, x(r)_{15}) \) at each round
  - First round, accessed table indices are \( x(i) = p_i \oplus k_i \) for all \( i = 0, \ldots, 15 \)
  - Recovering accessed table indices \( \Rightarrow \) recovering the key
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
    \[
    x_i^{(0)} = p_i \oplus k_i \quad \text{for all } i = 0, \ldots, 15
    \]
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

\[
x_i^{(0)} = p_i \oplus k_i \quad \text{for all } i = 0, \ldots, 15
\]

→ Recovering accessed table indices ⇒ recovering the key
Case Study #3: Attacking Cryptographic Algorithms

- Bouncy Castle → default implementation uses T-Tables
- Bouncy Castle → default implementation uses T-Tables
  → Let’s monitor which T-Table entry is accessed!
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
• 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!

![Address vs Plaintext byte values graph]
Case study #4
Monitoring ARM TrustZone
Case Study #4: Monitoring ARM TrustZone

- Hardware-based security technology

Information from the trusted world should not be leaked to the non-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
- 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

- 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

- 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

Information from the trusted world should not be leaked to the non-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
  - 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
• Library to build cross-platform cache attacks
• 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
All described examples have been developed on top of libflush
- All described examples have been developed on top of libflush
- Includes eviction strategies for several devices
• All described examples have been developed on top of libflush
• Includes eviction strategies for several devices
• Comes with example code and documentation
• 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

println 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

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