Side-Channel Lab I

Michael Schwarz

Security Week Graz 2019
Scope

• everyday hardware: servers, workstations, laptops, smartphones...
• remote side-channel attacks
Scope

• everyday hardware: servers, workstations, laptops, smartphones...
• remote side-channel attacks
Scope

- everyday hardware: servers, workstations, laptops, smartphones...
• everyday hardware: servers, workstations, laptops, smartphones...

• remote side-channel attacks
Side channels

- safe software infrastructure \rightarrow no bugs, e.g., Heartbleed
• **safe software** infrastructure → no bugs, e.g., Heartbleed
• does not mean safe execution
• **safe software** infrastructure → no bugs, e.g., Heartbleed
• does not mean safe execution
• information **leaks** because of the **hardware** it runs on
Side channels

- **safe software** infrastructure → no bugs, e.g., Heartbleed
- does not mean safe execution
- information **leaks** because of the **hardware** it runs on
- no “bug” in the sense of a mistake → lots of performance optimizations
Side channels

- **safe software** infrastructure $\rightarrow$ no bugs, e.g., Heartbleed
- does not mean safe execution
- information **leaks** because of the **hardware** it runs on
- no “bug” in the sense of a mistake $\rightarrow$ lots of performance optimizations

$\rightarrow$ crypto and sensitive info., e.g., keystrokes and mouse movements
Shared hardware

- Memory
  - Memory deduplication
  - Memory bus
  - DRAM row buffer

- x86 CPU
  - Branch prediction unit
  - Arithmetic logic unit
  - Data and instruction cache

Michael Schwarz — Security Week Graz 2019
Why targeting the cache?

- shared across cores
Why targeting the cache?

- shared across cores
- fast
Why targeting the cache?

- shared across cores
- fast

→ fast cross-core attacks!
Timing differences

- Caches improve performance

SRAM is expensive → small caches

Different timings for memory accesses
- Data is cached → cache hit → fast
- Data is not cached → cache miss → slow
Timing differences

- caches improve performance
- SRAM is expensive → small caches
Timing differences

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
Timing differences

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  - data is **cached** → cache hit → **fast**
Timing differences

- caches improve performance
- SRAM is expensive → small caches
- different timings for memory accesses
  - data is **cached** → cache hit → **fast**
  - data is **not cached** → cache miss → **slow**
Memory Hierarchy

- L1 and L2 are private
- Last-level cache is divided into slices
- Shared across cores
- Inclusive cache
- L1 and L2 are private
- Last-level cache is divided into slices
- Slices are shared across cores
- Inclusive cache design

Diagram:
- Core 0 with Registers, Buffers, L1, L2
- Core 1 with Registers, Buffers, L1, L2
- Core 2 with Registers, Buffers, L1, L2
- Core 3 with Registers, Buffers, L1, L2
- Ring bus connects cores
- LLC slices 0, 1, 2, 3
- DRAM

Michael Schwarz — Security Week Graz 2019
Memory Hierarchy

- L1 and L2 are private
- Last-level cache: divided into slices
- Shared across cores
- Inclusive
Memory Hierarchy

- L1 and L2 are private
• L1 and L2 are private
• last-level cache:
- L1 and L2 are private
- last-level cache:
  - divided in slices
- L1 and L2 are private
- last-level cache:
  - divided in **slices**
  - **shared** across cores
• L1 and L2 are private
• last-level cache:
  • divided in slices
  • shared across cores
  • inclusive
Inclusive property

- **inclusive** LLC: superset of L1 and L2

- data evicted from the LLC is also evicted from L1 and L2

- a core can evict lines in the private L1 of another core
Inclusive property

- **inclusive** LLC: superset of L1 and L2
- **inclusive** LLC: superset of L1 and L2
Inclusive property

- inclusive LLC: superset of L1 and L2

- data evicted from the LLC is also evicted from L1 and L2

- a core can evict lines in the private L1 of another core
Inclusive property

- **inclusive** LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
Inclusive property

- **inclusive** LLC: superset of L1 and L2
- data evicted from the LLC is also evicted from L1 and L2
- a core can **evict lines** in the private L1 of another core
On current Intel CPUs:

- Registers: 0-1 cycle
Latency comparison

On current Intel CPUs:

- Registers: 0-1 cycle
- L1 cache: 4 cycles
On current Intel CPUs:

- Registers: 0-1 cycle
- L1 cache: 4 cycles
- L2 cache: 12 cycles
Latency comparison

On current Intel CPUs:

- Registers: 0-1 cycle
- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles
Latency comparison

On current Intel CPUs:

- Registers: 0-1 cycle
- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles
- DRAM memory: >120 cycles
How every timing attack works:

- learn timing of different corner cases
Measuring timing leakage

How every timing attack works:

- learn timing of different corner cases
- later, we recognize these corner cases by timing only
1. build two cases: cache hits and cache misses
1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
4. find a threshold to distinguish the two cases
Cache hits

Loop:

1. measure time
Cache hits

Loop:

1. measure time
2. access variable (always cache **hit**)

Michael Schwarz — Security Week Graz 2019
Cache hits

Loop:

1. measure time
2. access variable (always cache **hit**)
3. measure time
Cache hits

Loop:

1. measure time
2. access variable (always cache **hit**)
3. measure time
4. update histogram with delta
Cache misses

Loop:

1. measure time
Cache misses

Loop:

1. measure time
2. access variable (always cache miss)
Cache misses

Loop:

1. measure time
2. access variable (always cache miss)
3. measure time
Loop:

1. measure time
2. access variable (always cache **miss**)
3. measure time
4. update histogram with delta
Cache misses

Loop:

1. measure time
2. access variable (always cache miss)
3. measure time
4. update histogram with delta
5. **flush** variable (clflush instruction)
Time to code
Accurate timings

- very short timings
- \texttt{rdtsc} instruction: cycle-accurate timestamps
Accurate timings

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

```c
[...]
rdtsc
function()
rdtsc
[...]
```
Accurate timings

- do you measure what you *think* you measure?
- **out-of-order** execution → what is really executed
Accurate timings

- do you measure what you *think* you measure?
- **out-of-order** execution → what is really executed

```
rdtsc
function()       rdtsc
[...]            rdtsc
rdtsc
function()       [...]```

Michael Schwarz — Security Week Graz 2019
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`
Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Timing differences

Access time [CPU cycles]

Number of accesses

- $10^1$
- $10^4$
- $10^7$

cache hits
Timing differences

Access time [CPU cycles]

Number of accesses

- **cache hits**
- **cache misses**

Michael Schwarz — Security Week Graz 2019
Find threshold

• as high as possible
Find threshold

- as high as possible
- most cache hits are below
Find threshold

• as high as possible
• most cache hits are below
• no cache miss below
• Hit → Data is fetched from buffers, L1, L2, or L3
Hit vs. Miss?

- Hit → Data is fetched from buffers, L1, L2, or L3
- Miss → Data is fetched from DRAM
From Histogram to Attack

Access time [CPU cycles]

Number of accesses

- cache hits
- cache misses

Michael Schwarz — Security Week Graz 2019
• cache attacks → exploit timing differences of memory accesses
Type of attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
Type of attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
Type of attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
Type of attacks

- cache attacks → exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- side-channel attack: one malicious process spies on benign processes
Type of attacks

- cache attacks $\rightarrow$ exploit timing differences of memory accesses
- attacker monitors which lines are accessed, not the content
- covert channel: two processes communicating with each other
  - not allowed to do so, e.g., across VMs
- side-channel attack: one malicious process spies on benign processes
  - e.g., steals crypto keys, spies on keystrokes
Flush+Reload

Shared Memory

ATTACKER
 flush
 access

VICTIM
 access
Flush+Reload

ATTACKER

flush
access

Shared Memory

cached

Shared Memory

cached

VICTIM

access

Michael Schwarz — Security Week Graz 2019
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access

Shared Memory
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER
flush
access

Shared Memory

VICTIM
access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

 Shared Memory

Victim accessed

Victim did not access

VICTIM

Michael Schwarz — Security Week Graz 2019
Signatures (RSA)

\[ M = C^d \mod n \]
Signatures (RSA)

\[ M = C^d \mod n \]

Result = C
Signatures (RSA)

\[ M = C^d \mod n \]

Result = Result \times Result \times C

square multiply
Signatures (RSA)

\[ M = C^d \mod n \]

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \ldots \\
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result}
\]

square
Signatures (RSA)

\[ M = C^d \mod n \]

\[ 1100110 \ldots \]

\[ \text{Result} = \text{Result} \times \text{Result} \quad \text{square} \]
Signatures (RSA)

\[ M = C^d \mod n \]

Result = Result × Result × C

\[ 1100110 \ldots \]

square multiply
Signatures (RSA)

\[ M = C^d \mod n \]

result = \text{result} \times \text{result} \times C
Signatures (RSA)

\[ M = C^d \mod n \]

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result}
\]

\underline{square}
Time to code
• locate key-dependent memory accesses
Side-channel attack on user input

- locate **key-dependent** memory accesses
- How to locate key-dependent memory accesses?
• It's complicated:
It's complicated:

- Large binaries and libraries (third-party code)
Challenges

- It's complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
Challenges

• It’s complicated:
  • Large binaries and libraries (third-party code)
  • Many libraries (gedit: 60MB)
  • Closed-source / unknown binaries
Challenges

• It's complicated:
  • Large binaries and libraries (third-party code)
  • Many libraries (gedit: 60MB)
  • Closed-source / unknown binaries
  • Self-compiled binaries
Challenges

- It's complicated:
  - Large binaries and libraries (third-party code)
  - Many libraries (gedit: 60MB)
  - Closed-source / unknown binaries
  - Self-compiled binaries

- Difficult to find **all** exploitable addresses
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically

Exploitation Phase
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)

Exploitation Phase
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”

Exploitation Phase
Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. "events" (keystrokes, encryptions, ...)
  - called "Cache Template"

Exploitation Phase

- Monitor exploitable addresses
Profiling Phase

Attacker address space

Cache

Victim address space

Cache is empty
Profiling Phase

Attacker address space

Attacker triggers an event

Victim address space
Profiling Phase

Attacker checks one address for cache hits ("Reload")
Profiling Phase

Update number of cache hits per event
Attacker flushes shared memory

Profiling Phase

Attacker address space

Cache

Victim address space

Shared 0x0

flush

Shared 0x0

Shared 0x0

Shared 0x0
Profiling Phase

Attacker address space

Shared 0x0

Cache

Victim address space

Shared 0x0

A

Repeat for higher accuracy
Profiling Phase

Attacker address space

Shared 0x40

Cache

Victim address space

Shared 0x40

Continue with next address
Profiling Phase

Attacker address space

Cache

Victim address space

Continue with next address

Michael Schwarz — Security Week Graz 2019
What to profile?

$> \text{ps -A | grep gedit} \\
$> \text{cat /proc/<pid>/maps}

00400000-00489000 r-xp 00000000 fd:01 396356 \\
/usr/bin/gedit

7f5a96991000-7f5a96a51000 r-xp 00000000 fd:01 399365 \\
/usr/lib/x86_64-linux-gnu/libgdk-3.so.0.2200.30

... \\

memory range, access rights, offset, –, –, file name
Profiling a single event

```
$> cd practicals/02_cache_template_attacks/
$> make
$> # start the targeted program (e.g., gedit)
$> sleep 2; ./profiling /usr/lib/x86_64-linux-gnu/libgdk-3.so.0.2200.30

... and hold down a key in the target program
```
Profiling a single event

```bash
$ cd practicals/02_cache_template_attacks/
$ make
$ # start the targeted program (e.g., gedit)
$ sleep 2; ./profiling /usr/lib/x86_64-linux-gnu/libgdk-3.so.0.2200.30
```

... and hold down a key in the target program
save addresses with peaks!
Exploitation phase

$> # ./spy <file> <offset>
$> ./spy /usr/lib/x86_64-linux-gnu/libgdk-3.so.0.2200.30 336896
Monitoring offset 336896
Hit #0
Hit #1
Hit #2
...

Time to code
% sleep 2;./spy 300 7f05140a4080-7f051417b000 r-xp 0x20000 08:02 268050
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so

shark% ./spy
**Profiling Phase: 1 Event, 1 Address**

**ADDRESS**

0x7c800

**KEY**

n
Example: Cache Hit Ratio for \((0x7c800, n)\): 200 / 200
Profiling Phase: All Events, 1 Address

ADDRESS

0x7c800

KEY

0 1 2 3 4 5 6 7 8 9 A B C D E F
Example: Cache Hit Ratio for (0x7c800, u): 13 / 200
Profiling Phase: All Events, 1 Address

Distinguish \( n \) from other keys by monitoring \( 0x7c800 \)
### Profiling Phase: All Events, All Addresses

<table>
<thead>
<tr>
<th>ADDRESS</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7c680</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>
<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>0x7c6c0</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>
<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>0x7c700</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>
<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>0x7c740</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>
<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>0x7c780</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>
<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>0x7c7c0</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>
<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>0x7c800</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>
<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>0x7c840</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>
<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>0x7c880</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>
<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>0x7c8c0</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>
<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>0x7c900</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>
<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>0x7c940</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>
<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>0x7c980</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>
<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>0x7c9c0</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>
<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>0x7ca00</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>
<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>0x7cb80</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>
<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>0x7cc40</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>
<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>0x7cc80</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>
<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>0x7ccc0</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>
<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>0x7cdd0</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>
<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>
Side-Channel Lab II

Michael Schwarz

Security Week Graz 2019
What is a covert channel?

- Two programs would like to communicate
What is a **covert channel**?

- Two programs would like to communicate but are *not allowed* to do so
What is a **covert channel**?

- Two programs would like to communicate but are **not allowed** to do so
  - either because there is no communication channel...
What is a *covert channel*?

- Two programs would like to communicate but are *not allowed* to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
What is a covert channel?

- Two programs would like to communicate but are not allowed to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
- Use side channels and stay stealthy
Covert channel
Covert channel
<table>
<thead>
<tr>
<th>method</th>
<th>raw capacity</th>
<th>err. rate</th>
<th>true capacity</th>
<th>env.</th>
</tr>
</thead>
<tbody>
<tr>
<td>F+F [Gru+16]</td>
<td>3968Kbps</td>
<td>0.840%</td>
<td>3690Kbps</td>
<td>native</td>
</tr>
<tr>
<td>F+R [Gru+16]</td>
<td>2384Kbps</td>
<td>0.005%</td>
<td>2382Kbps</td>
<td>native</td>
</tr>
<tr>
<td>E+R [Lip+16]</td>
<td>1141Kbps</td>
<td>1.100%</td>
<td>1041Kbps</td>
<td>native</td>
</tr>
<tr>
<td>P+P [Mau+17]</td>
<td>601Kbps</td>
<td>0.000%</td>
<td>601Kbps</td>
<td>native</td>
</tr>
<tr>
<td>P+P [Liu+15]</td>
<td>600Kbps</td>
<td>1.000%</td>
<td>552Kbps</td>
<td>virt</td>
</tr>
<tr>
<td>P+P [Mau+17]</td>
<td>362Kbps</td>
<td>0.000%</td>
<td>362Kbps</td>
<td>native</td>
</tr>
</tbody>
</table>
Sending Data (easy but inefficient)

Sender

...  
D (0x44)  
E (0x45)  
F (0x46)  
G (0x47)  
H (0x48)  
I (0x49)  
...  

Last-level cache

Cache Line on Page #0x43  
Cache Line on Page #0x44  
Cache Line on Page #0x45  
Cache Line on Page #0x46  
Cache Line on Page #0x47  
Cache Line on Page #0x48  
Cache Line on Page #0x49  
Cache Line on Page #0x4A  

Receiver
Sending Data (easy but inefficient)

Sender

... 
D (0x44)  
E (0x45)  
F (0x46)  
G (0x47)  
H (0x48)  
I (0x49)  
...

Last-level cache

Cache Line on Page #0x43
Cache Line on Page #0x44
Cache Line on Page #0x45
Cache Line on Page #0x46
Cache Line on Page #0x47
Cache Line on Page #0x48
Cache Line on Page #0x49
Cache Line on Page #0x4A

Receiver

flush
flush
flush
flush
flush
flush
flush
flush
flush
Sending Data (easy but inefficient)

Sender

... 
D (0x44) 
E (0x45) 
F (0x46) 
G (0x47) 
H (0x48) 
I (0x49) 
... 

Receiver

Last-level cache

Cache Line on Page #0x43 
Cache Line on Page #0x44 
Cache Line on Page #0x45 
Cache Line on Page #0x46 
Cache Line on Page #0x47 
Cache Line on Page #0x48 
Cache Line on Page #0x49 
Cache Line on Page #0x4A
Sending Data (easy but inefficient)

... 
D (0x44)  
E (0x45)  
F (0x46)  
**G (0x47)**  
H (0x48)  
I (0x49)  
...
Sending Data (easy but inefficient)

Sender

... 
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
... 

Receiver

Last-level cache

Cache Line on Page #0x43
Cache Line on Page #0x44
Cache Line on Page #0x45
Cache Line on Page #0x46
Cache Line on Page #0x47
Cache Line on Page #0x48
Cache Line on Page #0x49
Cache Line on Page #0x4A
Sending Data (easy but inefficient)

Sender

... 
D (0x44) 
E (0x45) 
F (0x46) 
G (0x47) 
H (0x48) 
I (0x49) 
... 

Receiver

Last-level cache

- Cache Line on Page #0x43
- Cache Line on Page #0x44
- Cache Line on Page #0x45
- Cache Line on Page #0x46
- Cache Line on Page #0x47
- Cache Line on Page #0x48
- Cache Line on Page #0x49
- Cache Line on Page #0x4A
sending data (easy but inefficient)

sender

... 
D (0x44)
E (0x45)
F (0x46)
G (0x47)
H (0x48)
I (0x49)
... 

receiver

last-level cache

- Cache Line on Page #0x43
- Cache Line on Page #0x44
- Cache Line on Page #0x45
- Cache Line on Page #0x46
- Cache Line on Page #0x47
- Cache Line on Page #0x48
- Cache Line on Page #0x49
- Cache Line on Page #0x4A

G (0x47)

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measure

measur
Sending Data (easy but inefficient)

Sender

... 
D (0x44) 
E (0x45) 
F (0x46) 
G (0x47) 
H (0x48) 
I (0x49) 
... 

Receiver

Last-level cache

Cache Line on Page #0x43
Cache Line on Page #0x44
Cache Line on Page #0x45
Cache Line on Page #0x46
Cache Line on Page #0x47
Cache Line on Page #0x48
Cache Line on Page #0x49
Cache Line on Page #0x4A
Sending Data (easy but inefficient)

... 
D (0x44)  
E (0x45)  
F (0x46)  
G (0x47)  
H (0x48)  
I (0x49)  
...
Sending Data (easy but inefficient)

...  
D (0x44)  
E (0x45)  
F (0x46)  
G (0x47)  
H (0x48)  
I (0x49)  
...  

Last-level cache

- Cache Line on Page #0x43
- Cache Line on Page #0x44
- Cache Line on Page #0x45
- Cache Line on Page #0x46
- Cache Line on Page #0x47
- Cache Line on Page #0x48
- Cache Line on Page #0x49
- Cache Line on Page #0x4A
Sending Data (easy but inefficient)

Sender

\[
\begin{align*}
\ldots \\
D & (0x44) \\
E & (0x45) \\
F & (0x46) \\
G & (0x47) \\
H & (0x48) \\
I & (0x49) \\
\ldots
\end{align*}
\]

Receiver

\[
\begin{align*}
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\text{measure} & \quad \text{measure} \\
\end{align*}
\]

Last-level cache

\[
\begin{align*}
\text{Cache Line on Page } & \#0x43 \\
\text{Cache Line on Page } & \#0x44 \\
\text{Cache Line on Page } & \#0x45 \\
\text{Cache Line on Page } & \#0x46 \\
\text{Cache Line on Page } & \#0x47 \\
\text{Cache Line on Page } & \#0x48 \\
\text{Cache Line on Page } & \#0x49 \\
\text{Cache Line on Page } & \#0x4A \\
\end{align*}
\]

Michael Schwarz — Security Week Graz 2019
Time to code
Operating Systems 101
• Kernel is isolated from user space
Memory Isolation

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
Memory Isolation

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel
• CPU support virtual address spaces to isolate processes
Paging

- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames
• CPU support **virtual address spaces** to isolate processes
• Physical memory is organized in **page frames**
• Virtual memory pages are **mapped** to page frames using page tables
Address Translation on x86-64

48-bit virtual address

PML4
PML4E 0
PML4E 1
...
#PML4I
...
PML4E 511

PDPT
PDPT 0
PDPT 1
...
#PDPTI
...
PDPT 511

Page Directory
PDE 0
PDE 1
...
PDE #PDI
...
PDE 511

Page Table
PTE 0
PTE 1
...
PTE #PTI
...
PTE 511

4 KiB Page
Byte 0
Byte 1
...
Offset
...
Byte 4095

Michael Schwarz — Security Week Graz 2019
Address Translation on x86-64

48-bit virtual address

PML4I (9 b)  PDPTI (9 b)  PDI (9 b)  PTI (9 b)  Offset (12 b)
### Page Table Entry

<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
</table>

- User/Supervisor bit defines in which privilege level the page can be accessed
- Kernel is typically mapped into every address space
Kernel is typically mapped into every address space.

Entire physical memory is mapped in the kernel.
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...).
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
• Serves as the interface between hardware and software
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
• Serves as the interface between hardware and software
• Microarchitecture is an actual implementation of the ISA
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
• Serves as the interface between hardware and software
• Microarchitecture is an actual implementation of the ISA
• Instructions are...
  • **fetched** (IF) from the L1 Instruction Cache
In-Order Execution

- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
In-Order Execution

• Instructions are...
  • fetched (IF) from the L1 Instruction Cache
  • decoded (ID)
  • executed (EX) by execution units
In-Order Execution

- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
  - executed (EX) by execution units
- Memory access is performed (MEM)
• Instructions are...
  • fetched (IF) from the L1 Instruction Cache
  • decoded (ID)
  • executed (EX) by execution units
• Memory access is performed (MEM)
• Architectural register file is updated (WB)
In-Order Execution

- Instructions are executed in-order
• Instructions are executed in-order
• Pipeline stalls when stages are not ready
In-Order Execution

- Instructions are executed **in-order**
- Pipeline **stalls** when stages are not ready
- If data is **not cached**, we need to wait
int width = 10, height = 5;

float diagonal = sqrt(width * width
                      + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
out-of-order execution

int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);

int area = width * height;

printf("Area %dx%d = %d\n", width, height, area);
Out-of-Order Execution

Instructions are

- fetched and decoded in the front-end
Instructions are

- fetched and decoded in the front-end
- dispatched to the backend
Out-of-Order Execution

Instructions are

- fetched and decoded in the front-end
- dispatched to the backend
- processed by individual execution units
Out-of-Order Execution

Instructions
- are executed out-of-order
Out-of-Order Execution

Instructions

- are executed out-of-order
- wait until their dependencies are ready
Out-of-Order Execution

Instructions

- are executed **out-of-order**
- wait until their **dependencies are ready**
  - Later instructions might execute prior earlier instructions
Out-of-Order Execution

Instructions

- are executed **out-of-order**
- wait until their **dependencies are ready**
  - Later instructions might execute prior earlier instructions
- retire **in-order**
Out-of-Order Execution

Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible
Out-of-Order Execution

Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible
- Exceptions are checked during retirement
Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
  - State becomes architecturally visible
- Exceptions are checked during retirement
  - Flush pipeline and recover state
The state does not become architecturally visible but . . .
The state does not become architecturally visible but . . .
Getting started...
Getting started...

- New code

```c
char data = 'S';  // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```
• New code

```c
char data = 'S'; // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```

• Luckily we know how to catch a segfault
Getting started...

- New code

```c
char data = 'S'; // a "secret" value
// ...
*(volatile char*) 0;
array[data * 4096] = 0;
```

- Luckily we know how to catch a segfault
- Then check whether any part of array is cached
Checking the array

- Flush+Reload over all pages of the array

![Access time vs. Page]

Access time [cycles]

Page
Time to code
• Add another *layer of indirection* to test

```c
char data = *(char *) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
Add another *layer of indirection* to test

```c
char data = *(char*) 0xfffffffff81a000e0;
array[data * 4096] = 0;
```
Which address?

- Check `/proc/kallsyms`

```
sudo cat /proc/kallsyms | grep banner
```
Which address?

- Check `/proc/kallsyms`

```
sudo cat /proc/kallsyms | grep banner
```

- or check `/proc/pid/pagemap` and print address

```
printf("target: %p\n",
    libsc_get_physical_address(ctx, vaddr));
```
Which address?

- Check `/proc/kallsyms`
  
  ```
  sudo cat /proc/kallsyms | grep banner
  ```

- or check `/proc/pid/pagemap` and print address
  
  ```
  printf("target: %p\n",
         libcsc_get_physical_address(ctx, vaddr));
  ```

- or start at a random address and iterate
Time to code
Spectre-PHT (aka Spectre Variant 1)

```plaintext
index = 0

if (index < 4) then
    glyph[data[index]]

else

Memory

data[0]
data[1]
data[2]
data[3]

...
index = 0

if (index < 4) {
    glyph[data[index]]
} else {
    Speculate
}

Michael Schwarz — Security Week Graz 2019
Spectre-PHT (aka Spectre Variant 1)

```
index = 0

if (index < 4)
    glyph[data[index]]
else
    {}
```
Spectre-PHT (aka Spectre Variant 1)

index = 0

Shared Memory

Execute

if (index < 4)

then

glyph[data[index]]

else

{}
index = 0

if (index < 4) then
  glyph[data[index]]
else
  {}

Shared Memory

Memory

DATA
KEY
...
Spectre-PHT (aka Spectre Variant 1)

**index = 1**

**Shared Memory**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

**Memory**

if (index < 4) then

$$\text{glyph}[\text{data}[\text{index}]]$$

else

{}
index = 1

if (index < 4)

then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

index = 1

if (index < 4)
then
glyph[data[index]]
else
{}

Shared Memory

Memory

A

D

B

E

C

F

D

G

E

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

index = 1

if (index < 4) {
    glyph[data[index]]
} else {
    }

Shared Memory

Speculate

Memory

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

\[
\text{index} = 1
\]

if (index < 4)

\[
\text{glyph}[\text{data}[\text{index}]]
\]

else

\[
\text{key}
\]

\[
\begin{align*}
\text{data}[0] \\
\text{data}[1] \\
\text{data}[2] \\
\text{data}[3]
\end{align*}
\]
index = 1

```
if (index < 4) {
  glyph[data[index]]
} else {
}
```
Spectre-PHT (aka Spectre Variant 1)

index = 2

Shared Memory

A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z

Speculate

if (index < 4)
then

glyph[data[index]]

else

Memory

{ }

D
data[0]
DATA
data[1]
TA
KEY
data[2]
KE
HY
data[3]
Y...
index = 2

if (index < 4)
then

glyph[data[index]] = T

else


index = 2

if (index < 4)
    glyph[data[index]]
else
    {}
Spectre-PHT (aka Spectre Variant 1)

index = 2

if (index < 4) {
    glyph[data[index]]
} else {

}
Spectre-PHT (aka Spectre Variant 1)

index = 2

```
index = 2

if (index < 4)
    glyph[data[index]]
else
    {}
```
Spectre-PHT (aka Spectre Variant 1)

index = 3

Shared Memory

Speculate

if (index < 4)

glyph[data[index]]

else

Memory

{ }

data[0]
data[1]
data[2]
data[3]
index = 3

if (index < 4) then

glyph[data[index]]

else

{}
index = 3

if (index < 4)
    glyph[data[index]]
else
    }

Shared Memory

glyph[data[index]]

Memory

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

index = 3

Shared Memory

Speculate

if (index < 4)

then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

index = 3

if (index < 4)
    glyph[data[index]]
else
    }

index = 3

Shared Memory

Execute

Memory

data[0]
data[1]
data[2]
data[3]
Spectre-PHT (aka Spectre Variant 1)

index = 4

if (index < 4)
   glyph[data[index]]
else
   {}

Shared Memory:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

Memory:

{ data[0], data[1], data[2], data[3] }

...
index = 4

Shared Memory

Speculate

if (index < 4)

glyph[data[index]]

else

Memory

index = 4

if (index < 4)

glyph[data[index]]

else

Memory

data[0]
data[1]
data[2]
data[3]

K

A

Michael Schwarz — Security Week Graz 2019
Spectre-PHT (aka Spectre Variant 1)

index = 4

if (index < 4) then

glyph[data[index]]

else

{}
Spectre-PHT (aka Spectre Variant 1)

index = 4

if (index < 4)
    glyph[data[index]]
else
    {}

Michael Schwarz — Security Week Graz 2019
Spectre-PHT (aka Spectre Variant 1)

```plaintext
index = 4

if (index < 4) then
  glyph[data[index]]
else
  {}
```

Shared Memory

```
A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
```

Memory

```
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
```

Execute

```
{}
operation #n
Spectre Root Cause

operation \#n

prediction

time
Spectre Root Cause

operation \#n

prediction

operation \#n+2

time
**Spectre Root Cause**

operation \( #n \)

prediction

operation \( #n+2 \)

possibly architectural transient execution

time

Michael Schwarz — Security Week Graz 2019
Spectre Root Cause

- Operation #n
- Retire
- Prediction
- Operation #n+2
- Transient execution
- Possibly architectural
- CF/DF
- Time
Spectre Root Cause

Operation \#n

Predict CF/DF

Possibly architectural transient execution

Flushing pipeline on wrong prediction

Operation \#n+2

Retire

Prediction

Retire

Time
Spectre Root Cause

operation #n
retire

predict
CF/DF

possibly architectural

transient execution

operation #n+2
retire

prediction

flush pipeline on wrong prediction
Many predictors in modern CPUs
Spectre Root Cause

- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
Many predictors in modern CPUs
- Branch taken/not taken (PHT)
- Call/Jump destination (BTB)
- Function return destination (RSB)
Spectre Root Cause

• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
  • Function return destination (RSB)
  • Load matches previous store (STL)
Many predictors in modern CPUs

- Branch taken/not taken (PHT)
- Call/Jump destination (BTB)
- Function return destination (RSB)
- Load matches previous store (STL)

Most are even shared among processes
Spectre Mistraining

same address space/
in place

Victim

branch
Spectre Mistraining

www.tugraz.at

same address space/
out of place

Victim

Congruent
branch

Address
collision

Victim
branch

same address space/
in place
Spectre Mistraining

same address space/
out of place

same address space/
in place

Victim

Congruent branch

Address collision

Victim branch

Shared Branch Prediction State
Spectre Mistraining

same address space/
out of place

same address space/
in place

Victim

Attacker

Congruent branch

Victim branch

Address collision

Shared Branch Prediction State

Michael Schwarz — Security Week Graz 2019
Spectre Mistraining

Victim

Same address space/
out of place

Congruent
branch

Address collision

Victim
branch

Same address space/
in place

Attacker

Shadow branch

cross address space/
in place

Shared Branch Prediction State

Michael Schwarz — Security Week Graz 2019
Spectre Mistraining

Victim

Congruent branch

Address collision

Victim branch

Attacker

Congruent branch

Address collision

Shadow branch

Shared Branch Prediction State

same address space/
out of place

cross address space/
out of place

same address space/
in place

cross address space/
in place
Time to code
Side-Channel Lab II

Michael Schwarz

Security Week Graz 2019

