Page Cache Attacks
Microarchitectural Attacks on Flawless Hardware

Daniel Gruss, Trishita Tiwari, Michael Schwarz, Erik Kraft
• Modern CPUs contain multiple microarchitectural elements
Modern CPUs contain multiple microarchitectural elements.
Modern CPUs contain multiple microarchitectural elements

- Caches and buffers
- Predictors

- Transparent for the programmer
Modern CPUs contain multiple microarchitectural elements

- Transparent for the programmer
- Timing optimizations → side-channel leakage
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

printf("%d", i);
printf("%d", i);

Cache miss
Request
Response

i

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
CPU Cache

printf("%d", i);

printf("%d", i);

Cache miss

Cache hit

Request

Response
CPU Cache

DRAM access, slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit
CPU Cache

DRAM access,
slow

printf("%d", i);

Cache miss

printf("%d", i);

Cache hit

No DRAM access,
much faster

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Caching speeds up Memory Accesses

Access time [CPU cycles]

Number of accesses

10^0 10^1 10^2 10^3 10^4 10^5 10^6

80 100 120 140 160 180 200 220 240 260 280 300 320 340 360 380 400

Cache Hits

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Caching speeds up Memory Accesses

![Graph showing cache hits and cache misses over access time]

Access time [CPU cycles]

Number of accesses

- Cache Hits
- Cache Misses

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload

Attacker

Shared Memory

Victim

flush
access

flush
access
Flush+Reload

Attacker
flush
access

Shared Memory

cached

Victim
access
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload

Attacker

`flush`

`access`

Shared Memory

Victim

`access`

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload

Attacker

flush
access

Shared Memory

Victim

access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload

Attacker

flush

access

Shared Memory

Victim

access
Flush+Reload

Attacker

flush

access

Victim

access

Shared Memory

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload

Attacker

flush

access

Shared Memory

Victim

access

Victim accessed (fast)

vs

Victim did not access (slow)

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker
prime
probe

Victim
access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker
prime
probe

Attacker Data

Victim
access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker
prime
probe

Attacker Data
Attacker Data
Attacker Data

Victim
access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Attacker Data
Attacker Data
Attacker Data
Attacker Data

Victim

access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Victim

access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker prime probe

Victim access

Attacker Data
Victim Data
Attacker Data
Attacker Data
Prime+Probe

Attacker
prime
probe

Victim
access

Attacker Data
Victim Data
Attacker Data
Attacker Data

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker data:
- Prime
- Probe

Victim data:
- Access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Victim

access

Attacker Data

Victim Data

Attacker Data

Attacker Data

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Attacker Data

Victim Data

Attacker Data

Attacker Data

Victim

access

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Victim

Attack Data

Victim Data

Attack Data

Attack Data

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker

prime

probe

Victim

Attacker Data

Victim Data

Attacker Data

Attacker Data

5

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Prime+Probe

Attacker prime probe vs Victim access

Attacker Data vs Victim Data
Attacker Data vs Victim Data
Attacker Data vs Victim Data

Victim did not access (fast)
Victim accessed (slow)

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Flush+Reload on Keystrokes

- Key presses trigger code execution in shared library (e.g., libgdk)
• Key presses trigger code execution in shared library (e.g., libgdk)
• Flush+Reload does not reveal actual key, only time difference between keys
Flush+Reload on Keystrokes

- Key presses trigger code execution in shared library (e.g., libgdk)
- Flush+Reload does not reveal actual key, only time difference between keys
- → Recover text with machine learning
Meltdown, Foreshadow, ZombieLoad, Spectre
Countermeasures for µ-arch. Attacks

- Deeply rooted in hardware → no real fixes
Countermeasures for \( \mu \)-arch. Attacks

- Deeply rooted in hardware \( \rightarrow \) no real fixes
- More isolation \( \rightarrow \) make exploitation harder
Countermeasures for $\mu$-arch. Attacks

- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder
- Attacks on design difficult to fix
Countermeasures for $\mu$-arch. Attacks

- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder
- Attacks on design difficult to fix
  - Caches $\rightarrow$ we want timing differences
Countermeasures for $\mu$-arch. Attacks

- Deeply rooted in hardware $\Rightarrow$ no real fixes
- More isolation $\Rightarrow$ make exploitation harder
- Attacks on design difficult to fix
  - Caches $\Rightarrow$ we want timing differences
  - Prediction $\Rightarrow$ we don’t want stalls
Countermeasures for $\mu$-arch. Attacks

- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder
- Attacks on design difficult to fix
  - Caches $\rightarrow$ we want timing differences
  - Prediction $\rightarrow$ we don’t want stalls
- So far: fixing symptoms
...in a parallel universe

Bugs are just in software, hardware is not so complex.

Yes, true.

Original image from commitstrip.com

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Thought experiment: what if there were no hardware side channels?
<table>
<thead>
<tr>
<th>µ-arch. Interface</th>
<th>Non-standard Syscall</th>
<th>ISA Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>CPU</td>
<td></td>
</tr>
</tbody>
</table>

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
# Operating System and CPU Microarchitecture

<table>
<thead>
<tr>
<th>μ-arch. Interface</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-standard Syscall</td>
<td>ISA Extension</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>μ-Architecture</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Caches</td>
<td>Hardware Caches</td>
</tr>
</tbody>
</table>

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
### Operating System and CPU Microarchitecture

<table>
<thead>
<tr>
<th>µ-arch. Interface</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-standard Syscall</td>
<td>ISA Extension</td>
</tr>
</tbody>
</table>

### µ-Architecture

<table>
<thead>
<tr>
<th>OS</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Caches</td>
<td>Hardware Caches</td>
</tr>
<tr>
<td>Software Prefetcher</td>
<td>Hardware Prefetcher</td>
</tr>
</tbody>
</table>

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Operating System and CPU Microarchitecture

μ-arch. Interface
- Non-standard Syscall
- ISA Extension

μ-Architecture
- Software Caches
- Hardware Caches
- Software Prefetcher
- Hardware Prefetcher

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
• Hardware → Software?
OS Side Channel

- Hardware $\rightarrow$ Software?
- Hardware-Agnostic Side Channel through the OS Page Cache
• Hardware → Software?
• Hardware-Agnostic Side Channel through the OS Page Cache
• Temporal resolution:
  - 2 µs (≤ 6.7 measurements per second)
  - 466 ns (≤ 223 measurements per second)
• Hardware → Software?
• Hardware-Agnostic Side Channel through the OS Page Cache
• Temporal resolution:
  2 µs (≤ 6.7 measurements per second)
  466 ns (≤ 223 measurements per second)
• Spatial resolution of 4 KiB
OS Side Channel

- Hardware → Software?
- Hardware-Agnostic Side Channel through the OS Page Cache
- Temporal resolution:
  - $2 \mu s \ (\leq 6.7 \text{ measurements per second})$
  - $466 \text{ ns} \ (\leq 223 \text{ measurements per second})$
- Spatial resolution of 4 KiB
- Various attacks: PHP RNG, UI-Redress, Windows ASLR, Keystroke Timings, Covert channels (local + remote)
- Managed by operating system
Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
- Implemented by all major operating systems
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

Attacker

Address Space

eviction#1
foo.so#1
eviction#2
foo.so#2
eviction#3
foo.so#3
eviction#4
foo.so#4

eviction#5

OS

Disk

Page Cache

RAM

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

faults

buffers foo.so#2

Disk

Disk

foo.so#2

Page Cache

RAM

eviction#1

eviction#2

eviction#3

eviction#4

eviction#5

foo.so#1
foo.so#2
foo.so#3
foo.so#4

Attacker

Accesses

Slow

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

Disk

Page Cache

foo.so#2

RAM

eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4

Attacker

Address Space

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

Disk

Page Cache

foo.so#2

RAM

eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4

Attacker

accesses

fast

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim
Address Space
foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

buffers eviction#5
fetches eviction#5

Page Cache
foo.so#2
eviction#5

RAM

faults

Disk

Attacker
Address Space

accesses

slow

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

Attacker

Address Space

eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

Disk

Page Cache

eviction#4
foo.so#2

buffers eviction#4

faults

fetches eviction#4

accesses

buffers eviction#4

slow

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

VMM

Disk

Page Cache

Page Cache

eviction#5
foo.so#2
eviction#3

RAM

Eviction #1
Eviction #2
Eviction #3
Eviction #4
Eviction #5

Attacker

Accesses

Fetches eviction#3

Buffers eviction#3

Faults

Slow

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

fetches eviction#2

Disk

Page Cache

buffers eviction#2

RAM

faults

Attacker

Accesses

Slow

foo.so#1
foo.so#2
foo.so#3
foo.so#4

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1
foo.so#2
foo.so#3
foo.so#4

OS

fetches eviction#1

buffers eviction#1

RAM

eviction#1
eviction#2
eviction#3
eviction#4
eviction#5

Page Cache

Disk

Accesses: eviction#1

Faults: eviction#1

Slow:

Attacker

accesses

buffers eviction#1

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim

Address Space

foo.so#1/foo.so#2
/foo.so#3
/foo.so#4

Disk

OS

Page Cache

eviction#5
eviction#4
eviction#3
eviction#2
eviction#1

RAM

Address Space

foo.so#1
/foo.so#2
/foo.so#3
/foo.so#4

Attacker

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Page Cache Attacks

Victim
Address Space

OS

Disk

Page Cache

RAM

Attacker
Address Space

buffers foo.so#2

fetches foo.so#2

faults
slow

accesses

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
I AM

WATCHING YOU
First idea:

- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:
- Use APIs provided by the operating system
  - mincore
  - QueryWorkingSetEx
- Non-destructive → higher average resolution
Observe Page Cache State

First idea:

- Measure page access time
Observe Page Cache State

First idea:

- Measure page access time
- Buffers pages in page cache → destructive
First idea:

- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution
Observe Page Cache State

First idea:

- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:
Observe Page Cache State

First idea:

- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:

- Use APIs provided by the operating system
Observe Page Cache State

First idea:
- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:
- Use APIs provided by the operating system
  - mincore
Observe Page Cache State

First idea:

- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:

- Use APIs provided by the operating system
  - mincore
  - QueryWorkingSetEx
Observe Page Cache State

First idea:
- Measure page access time
- Buffers pages in page cache → destructive
- Eviction always necessary → lower average resolution

Better:
- Use APIs provided by the operating system
  - mincore
  - QueryWorkingSetEx
- Non-destructive → higher average resolution
Reset Page Cache State

- Necessary for detecting multiple accesses
Reset Page Cache State

- Necessary for detecting **multiple** accesses
- **Bottleneck** of side channel
Reset Page Cache State

- Necessary for detecting **multiple** accesses
- **Bottleneck** of side channel
- Ideal strategy depends on memory management implementation
• Necessary for detecting multiple accesses
• Bottleneck of side channel
• Ideal strategy depends on memory management implementation
  • Differences in page replacement
• Necessary for detecting **multiple** accesses
• **Bottleneck** of side channel
• Ideal strategy depends on memory management implementation
  • Differences in **page replacement**
  • Global CLOCK-Pro like algorithm
• Necessary for detecting **multiple** accesses
• **Bottleneck** of side channel
• Ideal strategy depends on memory management implementation
  • Differences in **page replacement**
  • Global CLOCK-Pro like algorithm
  • Per-process working sets with Aging algorithm

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
- Access pages until target page is replaced
• Access pages until target page is replaced
• Basic eviction set: Large memory-mapped file
• Access pages until target page is replaced
• Basic eviction set: Large memory-mapped file
• -01: Add pages already in page cache
Eviction Linux

- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- `-01`: Add pages *already* in page cache
- `-02`: Fill memory with *anonymous dirty pages*
• Access pages until target page is replaced
• Basic eviction set: Large memory-mapped file
• -01: Add pages already in page cache
• -02: Fill memory with anonymous dirty pages
• Average run time down to 149 ms depending on optimisations
Eviction Windows

- Page cache eviction ↔ target page drops out of all working sets
Eviction Windows

- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
- Previous approach slow...
Eviction Windows

- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Optimizations:
  - Increase ws size + memory pressure $\rightarrow$ self-eviction ($<2$ s)
Eviction Windows

- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...

- Optimizations:
  - Increase ws size + memory pressure $\rightarrow$ self-eviction ($<2$ s)
  - Evicting any page in other processes
    $\rightarrow$ SetProcessWorkingSetSize (4.48 ms)
Eviction Windows

- Page cache eviction ↔ target page drops out of all working sets
  - Previous approach slow...

- Optimizations:
  - Increase ws size + memory pressure → **self-eviction** (<2 s)
  - Evicting any page in *other* processes
    → SetProcessWorkingSetSize (4.48 ms)
      - for processes with same integrity level as attacker
Eviction Windows

- Page cache eviction ↔ target page drops out of all working sets
  - Previous approach slow...
- Optimizations:
  - Increase ws size + memory pressure → self-eviction (<2 s)
  - Evicting any page in other processes
    → SetProcessWorkingSetSize (4.48 ms)
    - for processes with same integrity level as attacker
  - Evicting pages you have in your own working set
    → VirtualUnlock (17.69 µs)
THIS IS NOT DOCUMENTED

WHY IS IT DOING THIS?
Covert Channel

- Shared file as information carrier
Covert Channel

- **Shared file** as information carrier
- File page *presence* in page cache $\leftrightarrow$ message bits
- **Shared file** as information carrier
- File page **presence** in page cache $\leftrightarrow$ message bits
- Additional file pages for transmission control
Covert Channel

- **Shared file** as information carrier
- File page **presence** in page cache ↔ message bits
- Additional file pages for transmission control

Different implementation approaches:
Covert Channel

- Shared file as information carrier
- File page presence in page cache ↔ message bits
- Additional file pages for transmission control

Different implementation approaches:

<table>
<thead>
<tr>
<th>OS</th>
<th>Eviction</th>
<th>Observation</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>like side channel</td>
<td>mincore</td>
<td>20.20 kB/s</td>
</tr>
<tr>
<td></td>
<td>madvise</td>
<td>mincore</td>
<td>81.16 kB/s</td>
</tr>
<tr>
<td></td>
<td>posix_fadvise</td>
<td>mincore</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>process WS</td>
<td>QueryWorkingSetEx (ShareCount)</td>
<td>100.11 kB/s</td>
</tr>
<tr>
<td></td>
<td>VirtualUnlock</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Low bit error rate for all approaches (down to 0.000 03 %)

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Covert Channel

- **Shared file** as information carrier
- **File page** presence in page cache $\leftrightarrow$ message bits
- **Additional file pages** for transmission control

**Different implementation approaches:**

<table>
<thead>
<tr>
<th>OS</th>
<th>Eviction</th>
<th>Observation</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>like side channel</td>
<td>mincore</td>
<td>20.20 kB/s</td>
</tr>
<tr>
<td></td>
<td>madvise</td>
<td>mincore</td>
<td>81.16 kB/s</td>
</tr>
<tr>
<td></td>
<td>posix_fadvise</td>
<td>mincore</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>process WS</td>
<td>QueryWorkingSetEx (ShareCount)</td>
<td>100.11 kB/s</td>
</tr>
<tr>
<td></td>
<td>VirtualUnlock</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Low** bit error rate for all approaches (down to 0.000 03 %)

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
- Targets seeding of PHP PRNG
- Targets\textit{ seeding} of PHP PRNG
- \textit{microtime} used as seed by some frameworks
• Targets seeding of PHP PRNG
• `microtime` used as seed by some frameworks
  • Returns UNIX timestamp in microseconds
PHP RNG Attack

- Targets **seeding** of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
• Targets **seeding** of PHP PRNG
• `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
• Later PRNG used for cryptographic operations :(
• Side channel used to detect `microtime` call
PHP RNG Attack

- Targets **seeding** of PHP PRNG
- **microtime** used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to **detect** microtime call
  - Seed recoverable

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :
- Side channel used to detect `microtime` call
  - Seed recoverable
- `zif_microtime` on page 781 of php-fpm executable
PHP RNG Attack

- Targets **seeding** of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to detect `microtime` call
  - Seed recoverable
- `zif_microtime on page 781 of php-fpm executable`
  - PHP 7.3.5, depends on build environment/configuration
PHP RNG Attack

- Targets **seeding** of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to **detect** `microtime` call
  - Seed recoverable
- `zif_microtime` on page 781 of `php-fpm` executable
  - PHP 7.3.5, depends on build environment/configuration
- Average detection accuracy: \( \pm 1 \text{ms} \)

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Live Demo
UI Redressing Attack

- Detect opening of interesting window
• Detect opening of interesting **window**
  • e.g. authentication windows
UI Redressing Attack

- Detect opening of interesting **window**
  - e.g. authentication windows
- Overlay original window with **fake**
UI Redressing Attack

- Detect opening of interesting **window**
  - e.g. authentication windows
- Overlay original window with **fake**
- Side channel used as a **trigger**
UI Redressing Attack

- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger
- Provides low latency → hardly noticeable

Tested with root authentication window on Ubuntu 16.04
UI Redressing Attack

- Detect opening of interesting **window**
  - e.g. authentication windows
- Overlay original window with **fake**
- Side channel used as a **trigger**
- Provides **low latency** → hardly noticeable
- Tested with root authentication window on Ubuntu 16.04
UI Redressing Attack

- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger
- Provides low latency → hardly noticeable
- Tested with root authentication window on Ubuntu 16.04
  - Page 6 of binary polkit-gnome-authentication-agent-1

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
UI Redressing Attack

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Impact

- Identified as CVE-2019-5489
Impact

- Identified as CVE-2019-5489
- Linux and Windows deployed countermeasures
Impact

- Identified as CVE-2019-5489
- Linux and Windows deployed countermeasures
PATCHES

PATCHES EVERYWHERE
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes

---

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
  - ShareCount hidden for unprivileged users
  - No indirect querying of working set state
  - No non-destructive probing of higher-integrity processes → weaker attack
  - If `QueryWorkingSetEx` only possible leakage source
  - Page-cache eviction already harder than on Linux
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
  - `ShareCount` hidden for unprivileged users

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
Countermeasures - Windows

- Higher privilege for QueryWorkingSetEx on other processes
  - No direct querying of working set state
- ShareCount hidden for unprivileged users
  - No indirect querying of working set state
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` hidden for unprivileged users
  - No indirect querying of working set state
- No non-destructive probing of higher-integrity processes
Countermeasures - Windows

- Higher **privilege** for QueryWorkingSetEx on other processes
  - No direct querying of working set state
- **ShareCount** hidden for unprivileged users
  - No indirect querying of working set state
- No non-destructive probing of higher-integrity processes
  → **weaker** attack
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` hidden for unprivileged users
  - No indirect querying of working set state
- No non-destructive probing of higher-integrity processes
  → weaker attack
  - If `QueryWorkingSetEx` only possible leakage source
Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` hidden for unprivileged users
  - No indirect querying of working set state
- No non-destructive probing of higher-integrity processes
  - Weaker attack
    - If `QueryWorkingSetEx` only possible leakage source
    - Page-cache eviction already harder than on Linux

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
• mincore only reveals information for writeable file pages
Countermeasures - Linux

• mincore only reveals information for writeable file pages
  • Read-only files excluded → shared libraries, executables

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)
• `mincore` only reveals information for writeable file pages
  • Read-only files excluded → shared libraries, executables
  • Merged with the release of the 5.1.4 kernel
• \texttt{mincore} only reveals information for \textit{writeable} file pages
  • Read-only files excluded $\rightarrow$ \textit{shared} libraries, executables
  • Merged with the release of the 5.1.4 kernel
• Non-destructive probing no longer possible?
Countermeasures - Linux

- `mincore` only reveals information for *writeable* file pages
  - Read-only files excluded → *shared* libraries, executables
  - Merged with the release of the 5.1.4 kernel
- Non-destructive probing no longer possible?
  - No, `preadv2` with `RWF_NOWAIT` leaks same information
Countermeasures - Linux

- `mincore` only reveals information for *writeable* file pages
  - Read-only files excluded → *shared* libraries, executables
  - Merged with the release of the 5.1.4 kernel
- Non-destructive probing no longer possible?
  - No, `preadv2` with `RWF_NOWAIT` leaks same information
  - Not fixed yet
Countermeasures are Difficult

- We want the performance optimizations
Countermeasures are Difficult

- We want the performance optimizations
- Many side-channel attacks exploit intended behavior
Countermeasures are Difficult

- We want the performance optimizations
- Many side-channel attacks exploit intended behavior
- Often a trade-off between security and performance
Countermeasures are Difficult

- We want the performance optimizations
- Many side-channel attacks exploit intended behavior
- Often a trade-off between security and performance
- Every optimization is potentially a side channel
The Future

- We won’t get rid of side channels
The Future

- We won’t get rid of side channels
- More optimizations → more side channels
The Future

- We won’t get rid of side channels
- More optimizations $\rightarrow$ more side channels
- More attacks on the “OS microarchitecture”
• Abstraction leads to side channels
Take Aways

- **Abstraction** leads to side channels
- Software-cache attacks are similar to hardware-cache attacks
• Abstraction leads to side channels
• Software-cache attacks are similar to hardware-cache attacks
• Finding countermeasures is difficult
We want to thank James Forshaw for helpful discussions on COM use cases and Simon Gunacker for early explorative work on this topic. Daniel Gruss and Michael Schwarz were supported by a generous gift from ARM and also by a generous gift from Intel. Ari Trachtenberg and Trishita Tiwari were supported, in part, by the National Science Foundation under Grant No. CCF-1563753 and Boston University’s Distinguished Summer Research Fellowship, Undergraduate Research Opportunities Program, and the department of Electrical and Computer Engineering. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.