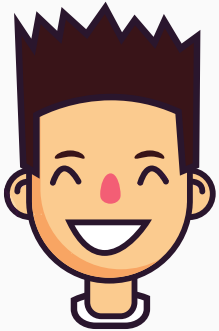
RuhrSec

Are Microarchitectural Attacks still possible on Flawless Hardware?

Michael Schwarz
Erik Kraft

Who am I?



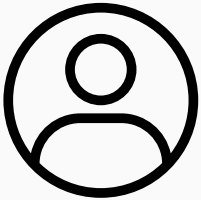
Michael Schwarz

PhD candidate @ Graz University of Technology

@misc0110

michael.schwarz@iaik.tugraz.at

Who am I?



Erik Kraft

Master student @ Graz University of Technology

@ekraft95

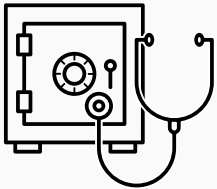
erik.kraft@student.tugraz.at

And the Team

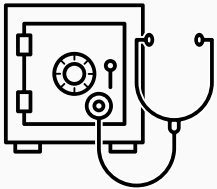
- Daniel Gruss (Graz University of Technology)
- Trishita Tiwari (Boston University)
- Ari Trachtenberg (Boston University)
- Jason Hennessey (NetApp)
- Alex Ionescu (CrowdStrike)
- Anders Fogh (Intel)

Side-Channel Attacks

- Bug-free software does not mean safe execution

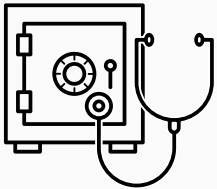


Side-Channel Attacks



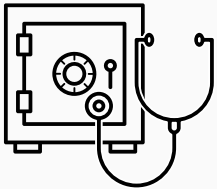
- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware

Side-Channel Attacks



- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

Side-Channel Attacks



- Bug-free software does not mean safe execution
- Information leaks due to underlying hardware
- Exploit leakage through side-effects

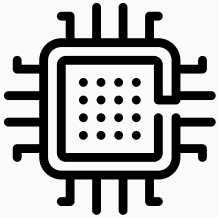
Power consumption

Execution time

CPU caches

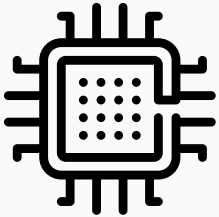
•••

Architecture and Microarchitecture



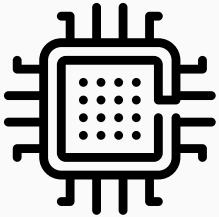
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)

Architecture and Microarchitecture



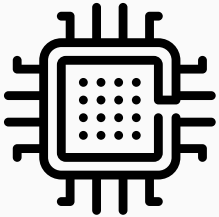
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software

Architecture and Microarchitecture



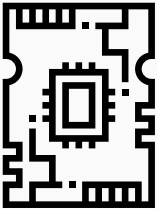
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

Architecture and Microarchitecture



- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . .)
- Interface between hardware and software
- Microarchitecture is an ISA implementation

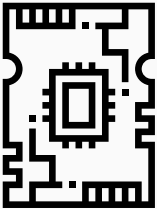
Microarchitectural Components



- Modern CPUs contain multiple microarchitectural elements

Microarchitectural Components

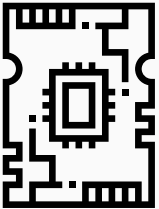
- Modern CPUs contain multiple microarchitectural elements



Caches and buffers

Predictors

Microarchitectural Components



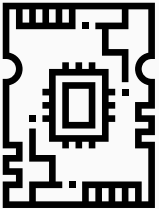
- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

- Transparent for the programmer

Microarchitectural Components



- Modern CPUs contain multiple microarchitectural elements

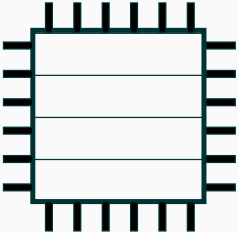
Caches and buffers

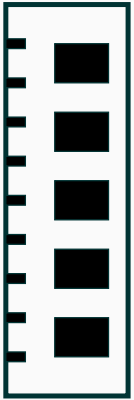
Predictors

- Transparent for the programmer
- Timing optimizations → side-channel leakage

CPU Cache

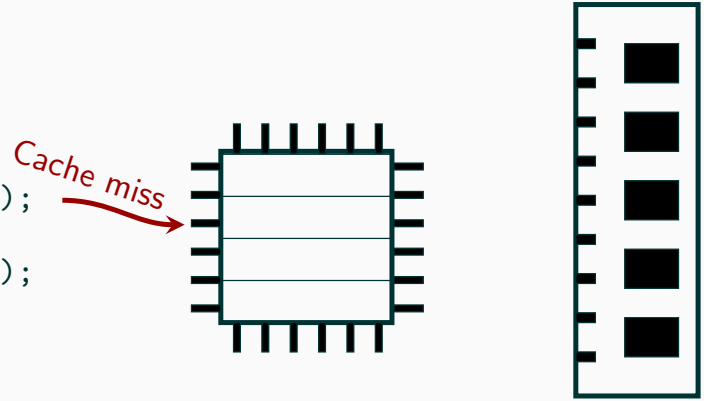
```
printf("%d", i);

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





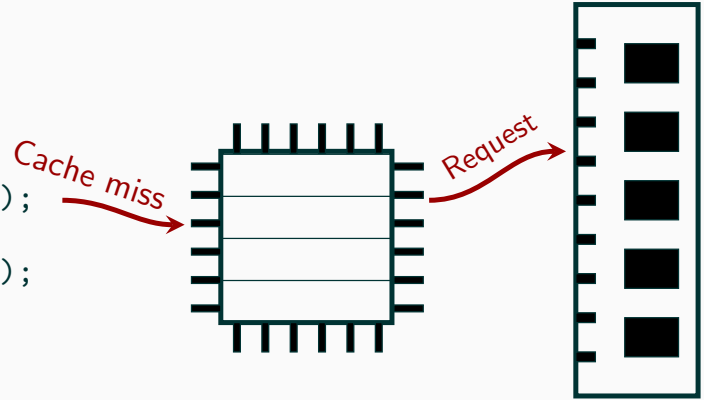
CPU Cache

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

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



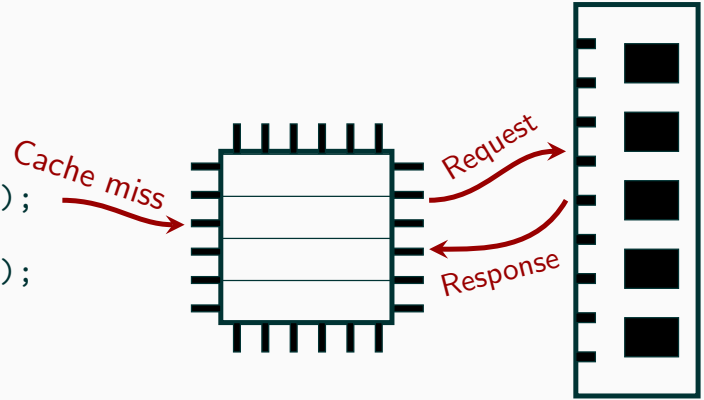
CPU Cache

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

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



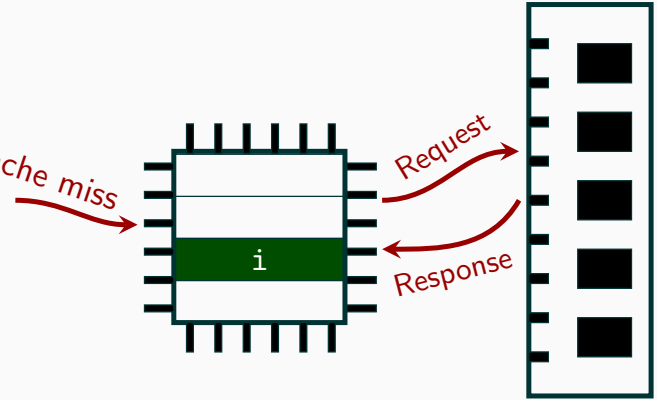
CPU Cache

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

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



CPU Cache

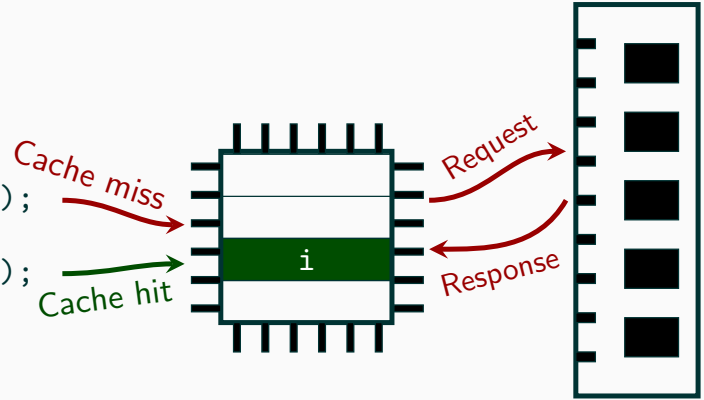
```
printf("%d", i);

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



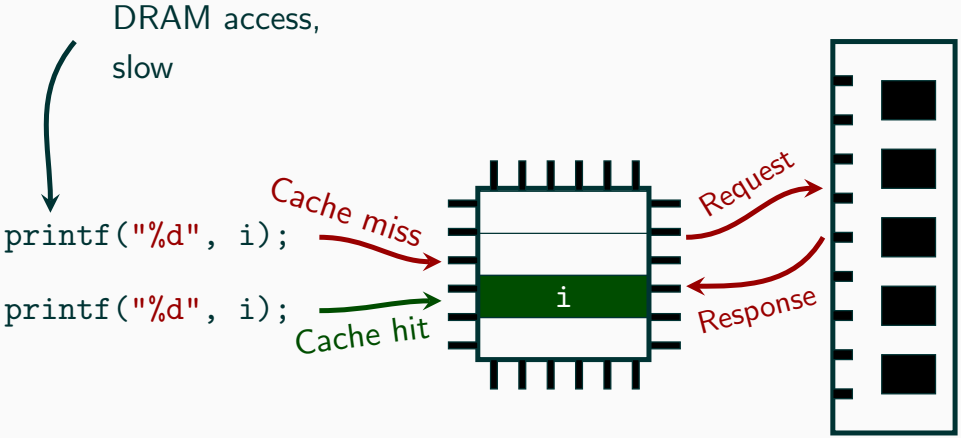
CPU Cache

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

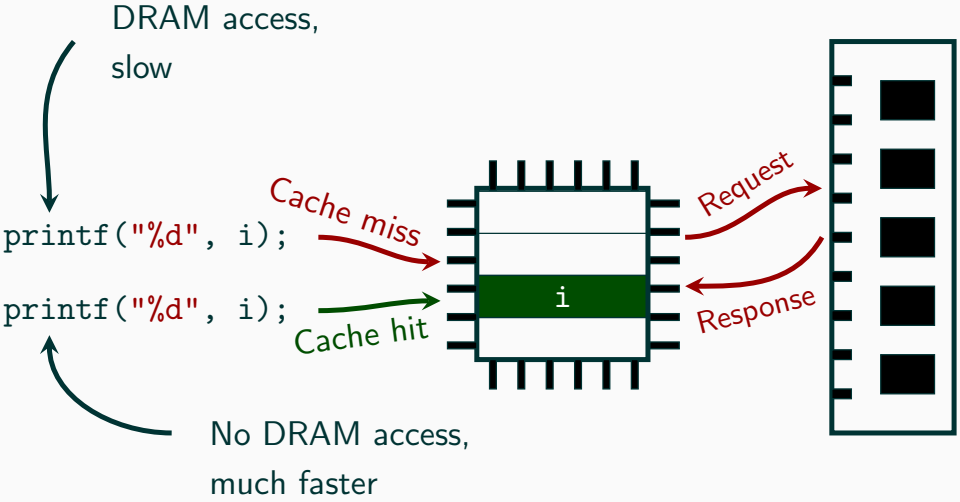
```
printf("%d", i);
```



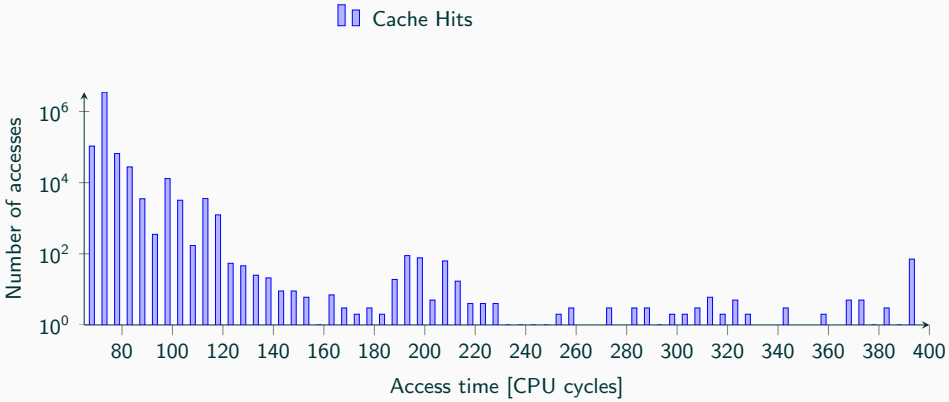
CPU Cache


DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response

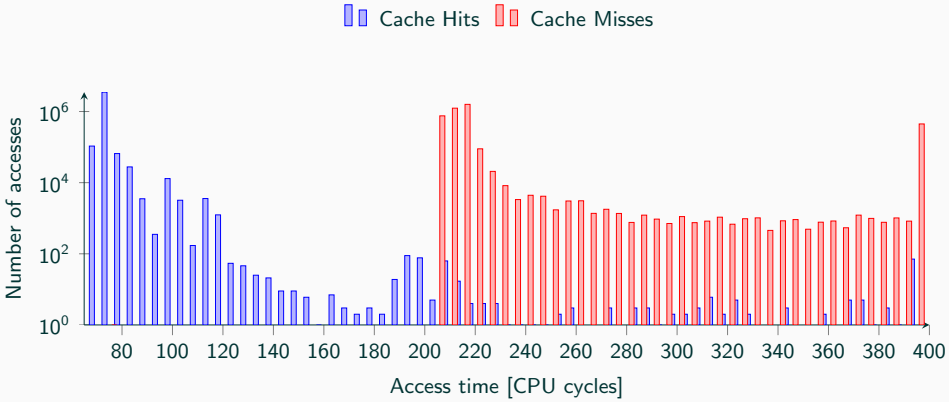
CPU Cache


DRAM access,
slow
printf("%d", i);
printf("%d", i);
Cache miss
Cache hit
i
Request
Response
No DRAM access,
much faster

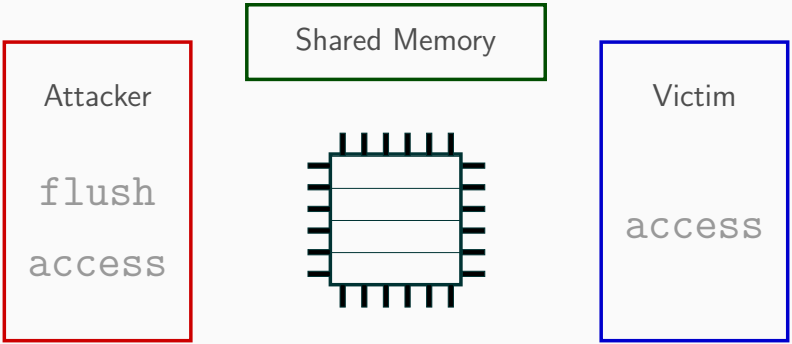
Caching speeds up Memory Accesses



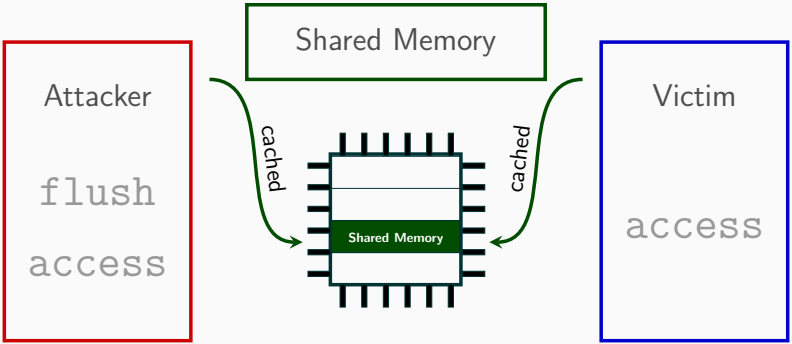
Caching speeds up Memory Accesses



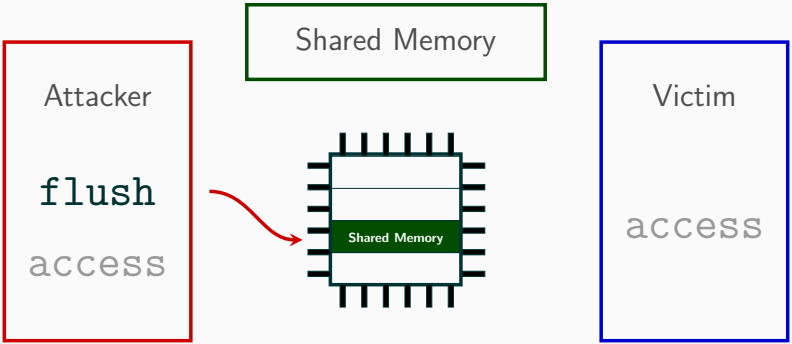
Flush+Reload



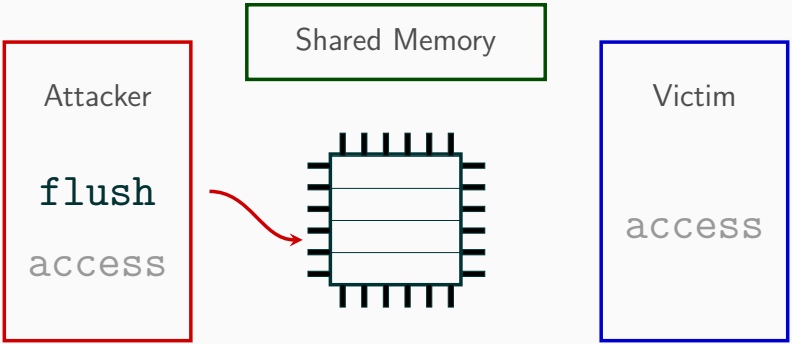
Flush+Reload



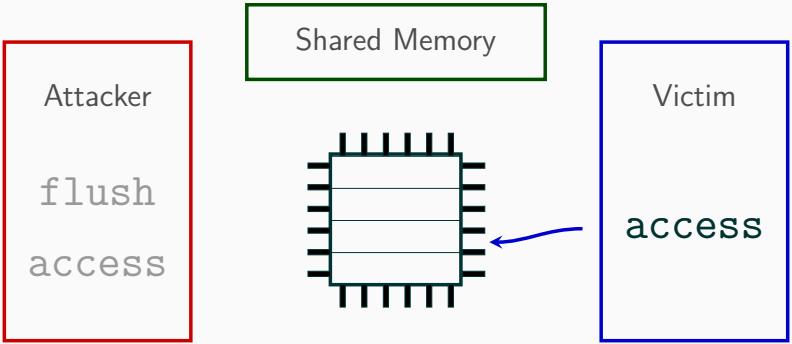
Flush+Reload



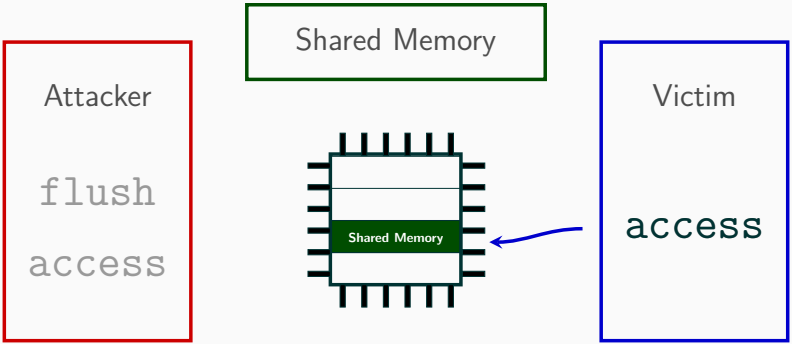
Flush+Reload



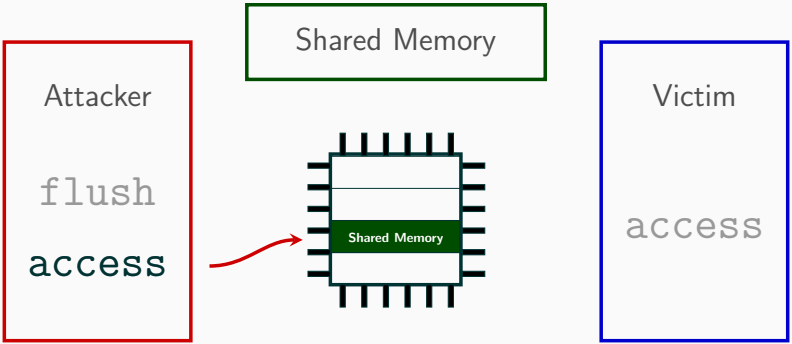
Flush+Reload



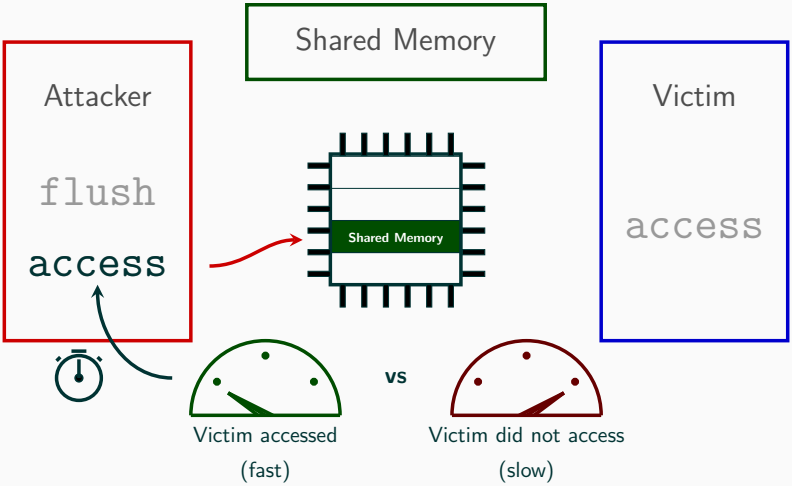
Flush+Reload



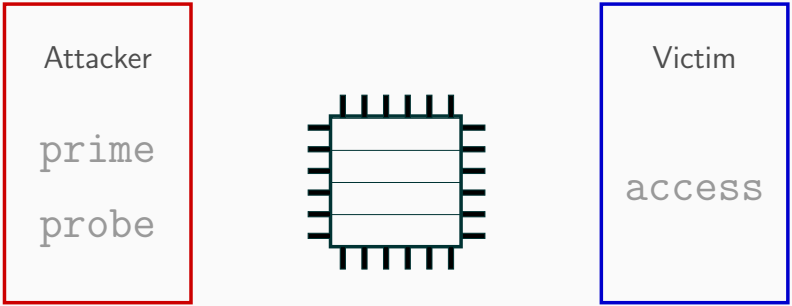
Flush+Reload



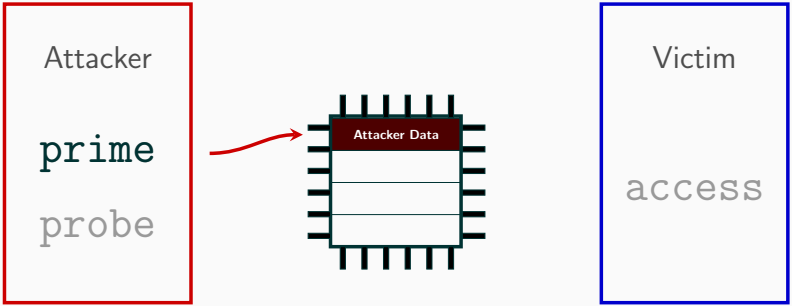
Flush+Reload



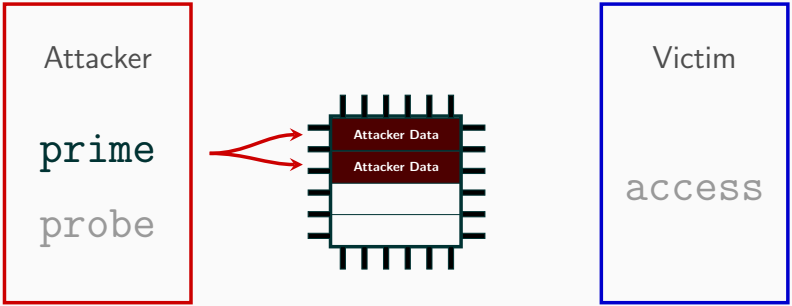
Prime+Probe



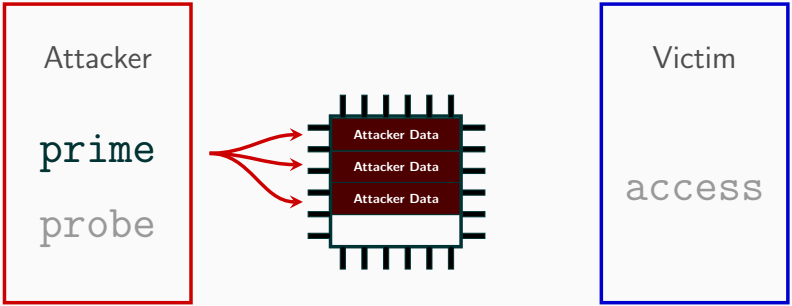
Prime+Probe



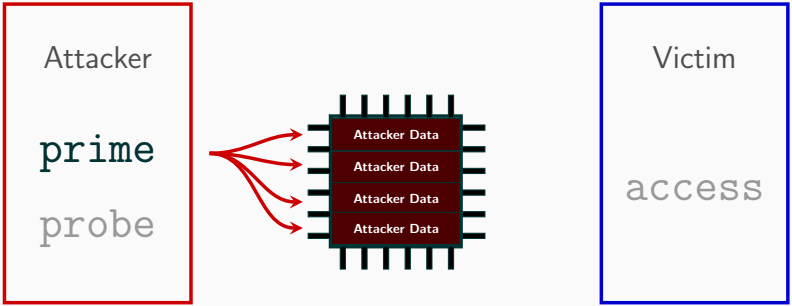
Prime+Probe


Attacker
prime
probe
Attacker Data
Attacker Data
Victim
access

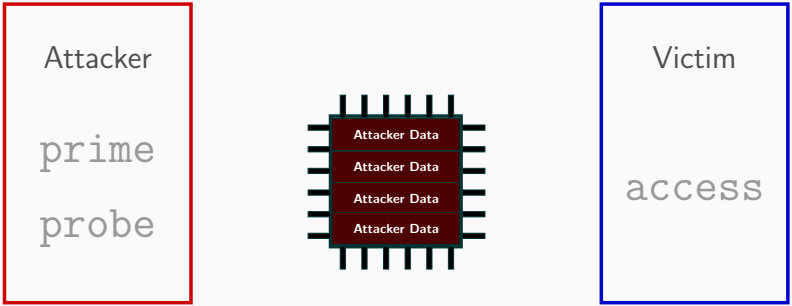
Prime+Probe


Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Victim
access

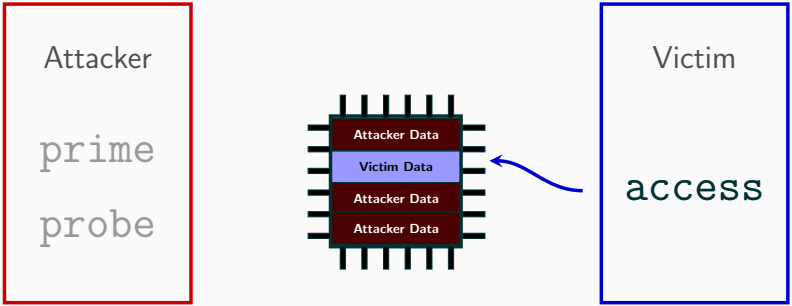
Prime+Probe


Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Attacker Data
Victim
access

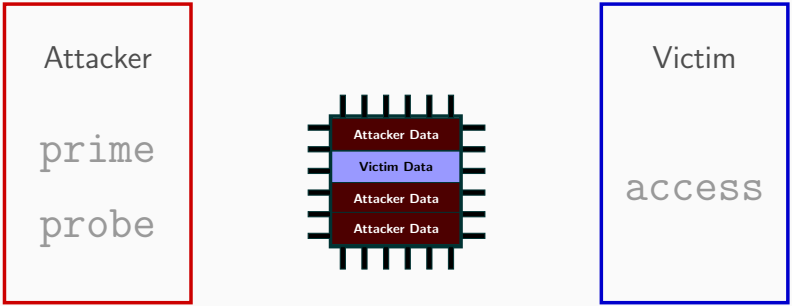
Prime+Probe


Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Attacker Data
Victim
access

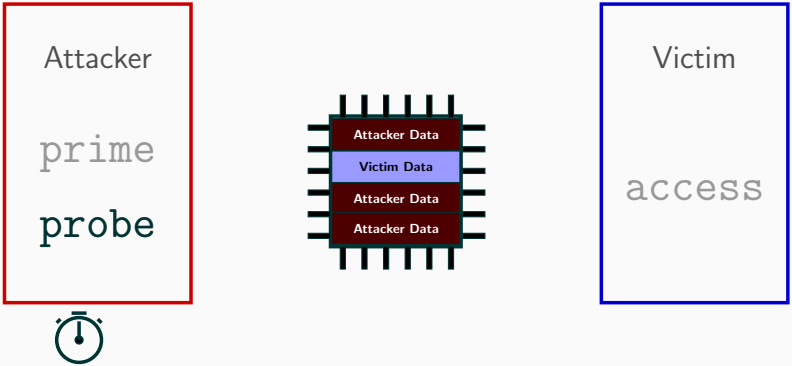
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

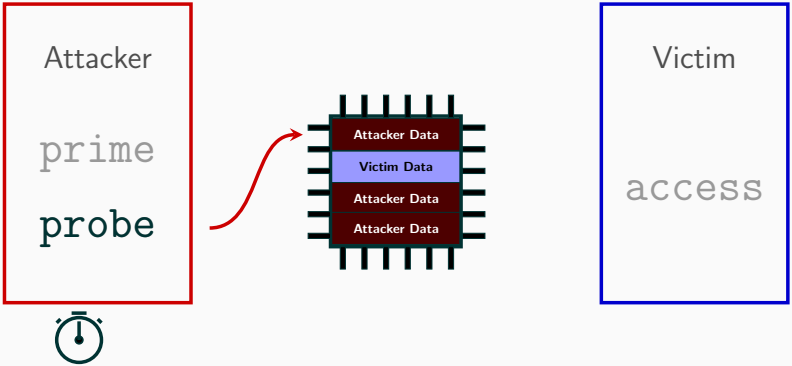
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

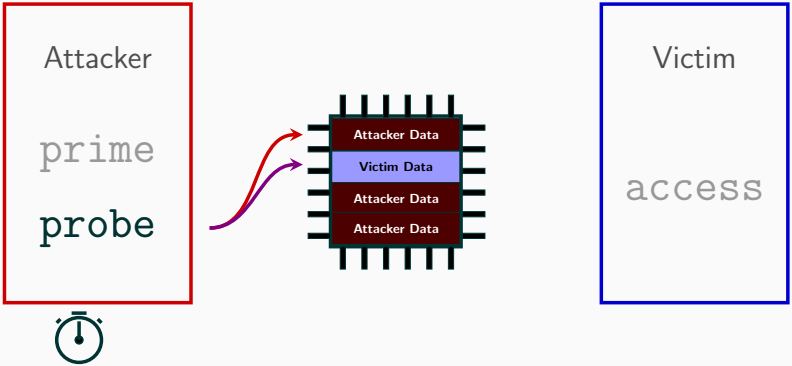
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

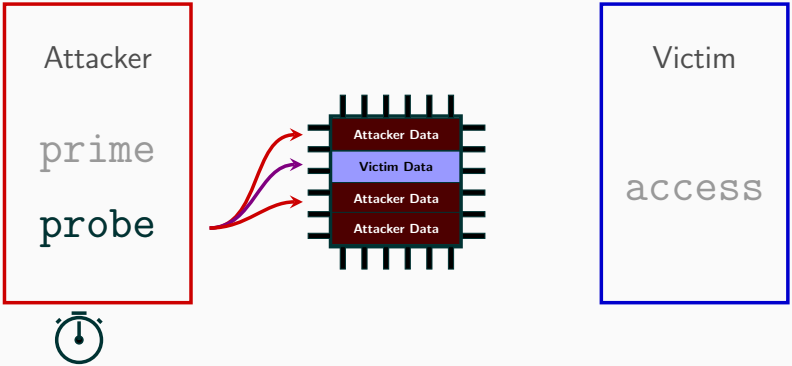
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

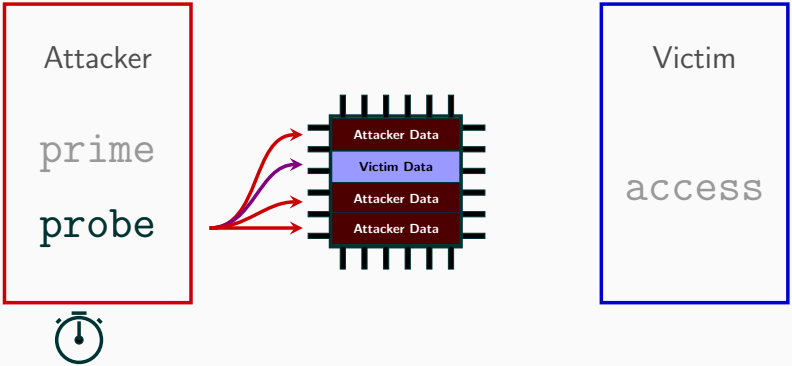
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

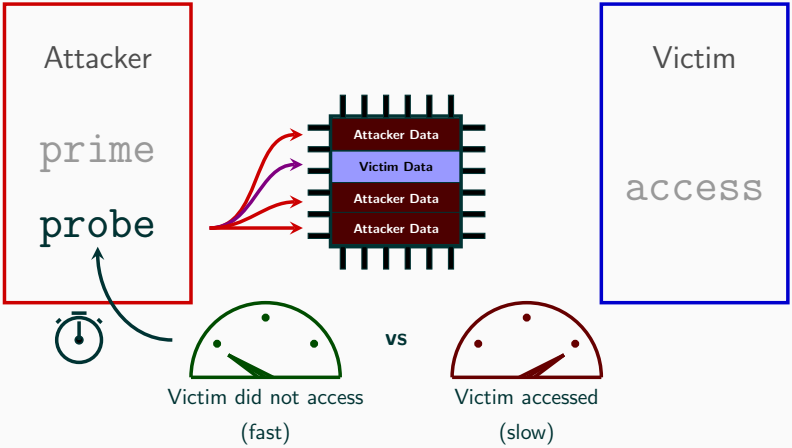
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

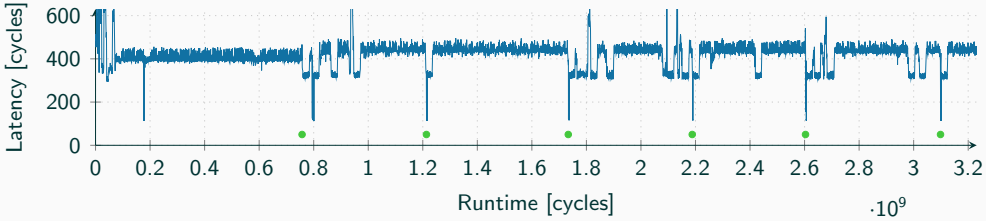
Prime+Probe


Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access

Prime+Probe

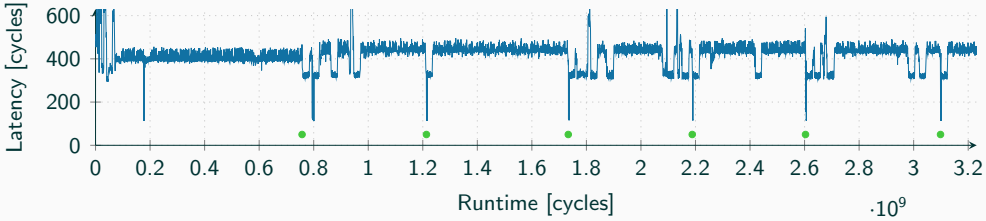

Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access
vs
Victim did not access
(fast)
Victim accessed
(slow)

Flush+Reload on Keystrokes



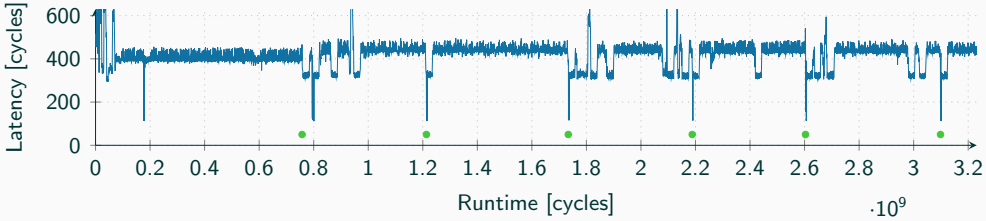
- Key presses trigger code execution in shared library (e.g., `libgdk`)

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

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
- $\rightarrow$ Recover text with machine learning

$$M = C^{d} \bmod n$$

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	···

Result	=	C

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	$\cdots$

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}} \times \underbrace{\boxed{C}}_{\text{multiply}}$$

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	$\cdots$

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}}$$

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	$\cdots$

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}}$$

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	$\cdots$

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}} \times \underbrace{\boxed{C}}_{\text{multiply}}$$

$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	···

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}} \times \underbrace{\boxed{C}}_{\text{multiply}}$$

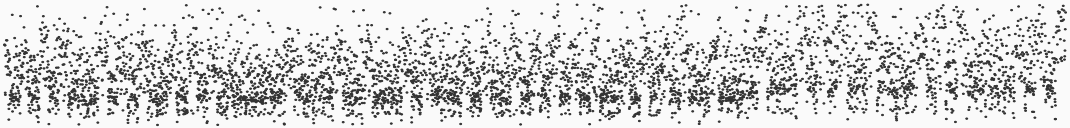
$$M = C^{\boxed{d}} \bmod n$$

1	1	0	0	1	1	0	···

$$\boxed{\text{Result}} = \underbrace{\boxed{\text{Result}} \times \boxed{\text{Result}}}_{\text{square}}$$

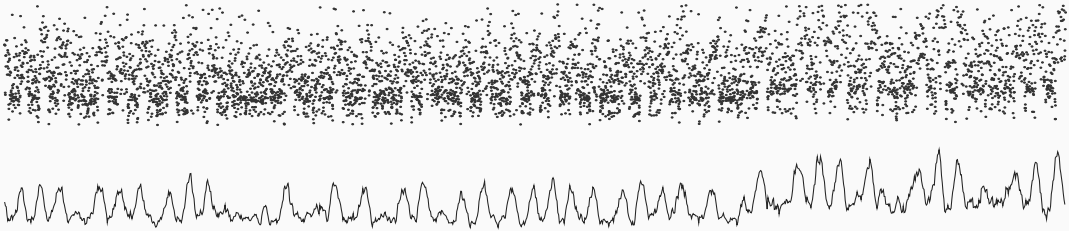
Measured Trace

Raw Prime+Probe trace...



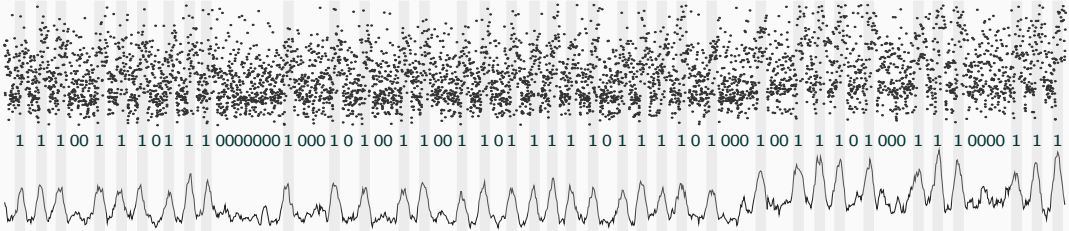
Measured Trace

...processed with a simple moving average...



Measured Trace

...allows to clearly see the bits of the exponent



Meltdown

- CPU vulnerability → out-of-order execution optimization

Meltdown

- CPU vulnerability → out-of-order execution optimization
- Deferred privilege check → access kernel memory

Meltdown

- CPU vulnerability → out-of-order execution optimization
- Deferred privilege check → access kernel memory
- Encode transiently leaked value via cache

Meltdown

- CPU vulnerability → out-of-order execution optimization
- Deferred privilege check → access kernel memory
- Encode transiently leaked value via cache
- Recover from cache using cache attack

Foreshadow

- Similar to Meltdown

Foreshadow

- Similar to Meltdown
- Forwarding non-present addresses to the L1 cache

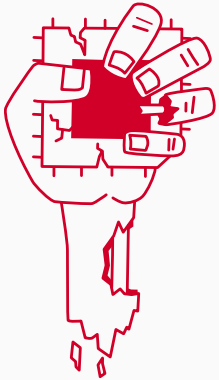
Foreshadow

- Similar to Meltdown
- Forwarding non-present addresses to the L1 cache
- Encode data leaked from L1 into the cache

Foreshadow

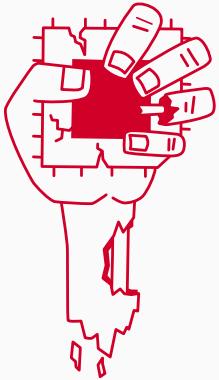
- Similar to Meltdown
- Forwarding non-present addresses to the L1 cache
- Encode data leaked from L1 into the cache
- Recover from cache using cache attack

ZombieLoad



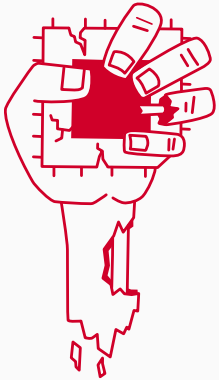
- Trigger complex memory-load situations

ZombieLoad



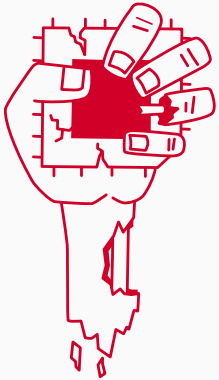
- Trigger complex memory-load situations
- CPU transiently forwards data from wrong locations

ZombieLoad



- Trigger complex memory-load situations
- CPU transiently forwards data from wrong locations
- Encode these values via cache

ZombieLoad

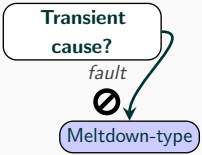


- Trigger complex memory-load situations
- CPU transiently forwards data from wrong locations
- Encode these values via cache
- Recover from cache using cache attack

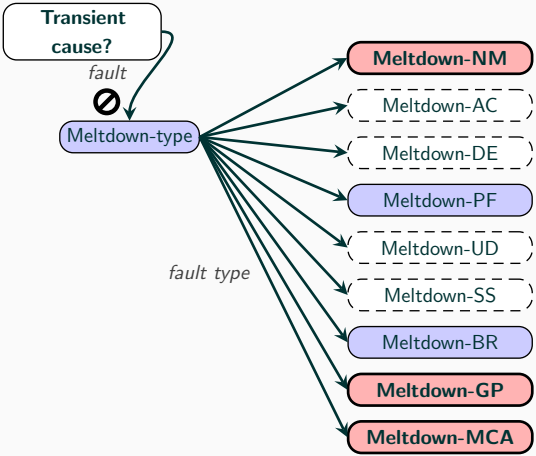
Meltdown Tree

Transient cause?

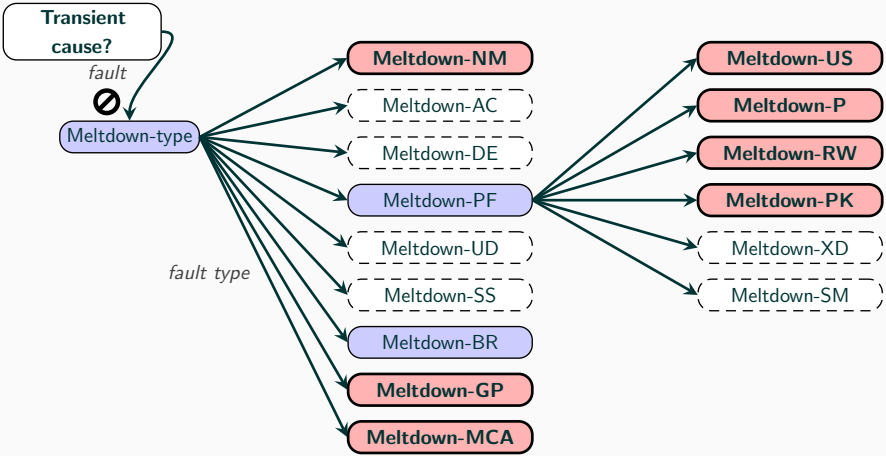
Meltdown Tree



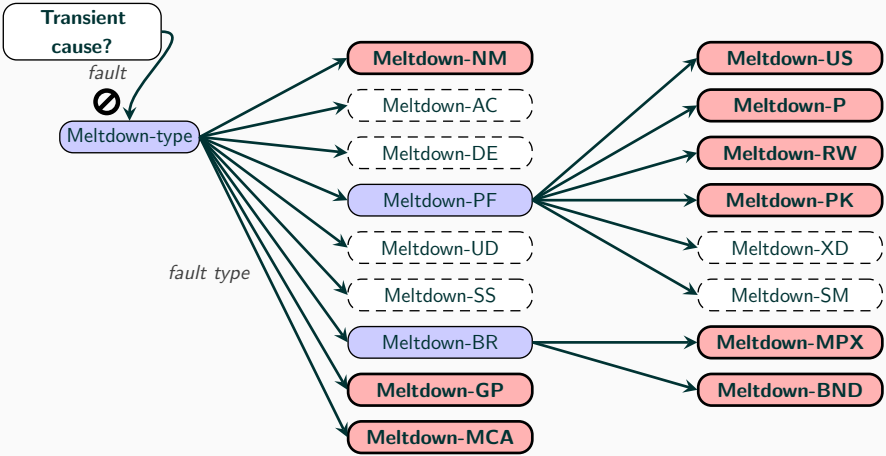
Meltdown Tree



Meltdown Tree



Meltdown Tree



Meltdown Outlook

- Meltdown is not a fully solved issue

Meltdown Outlook

- Meltdown is not a fully solved issue
- The tree is extensible

Meltdown Outlook

- Meltdown is not a fully solved issue
- The tree is extensible
- More Meltdown-type issues to come

Spectre

- Mistrain CPUs internal predictors

Spectre

- Mistrain CPUs internal predictors
- CPU speculatively works with unintended values

Spectre

- Mistrain CPUs internal predictors
- CPU speculatively works with unintended values
- Encode these values via cache

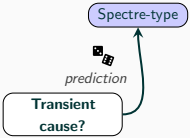
Spectre

- Mistrain CPUs internal predictors
- CPU speculatively works with unintended values
- Encode these values via cache
- Recover from cache using cache attack

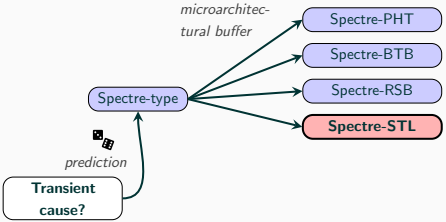
Spectre Variants

Transient cause?

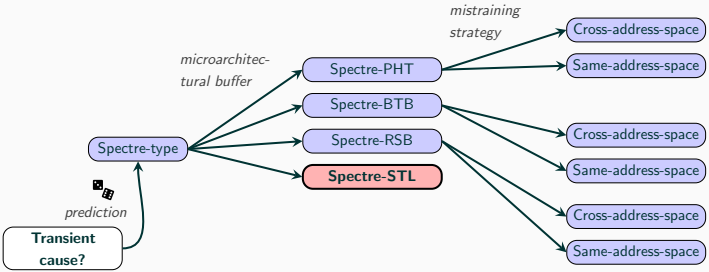
Spectre Variants



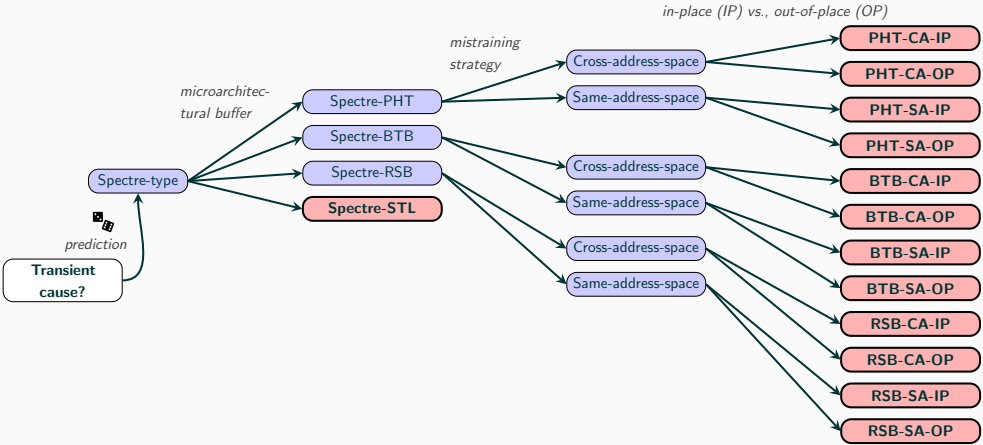
Spectre Variants



Spectre Variants



Spectre Variants



Countermeasures for μ-arch. Attacks

- Deeply rooted in hardware $\rightarrow$ no real fixes

Countermeasures for μ-arch. Attacks

- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder

Countermeasures for μ-arch. Attacks

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

Countermeasures for μ-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

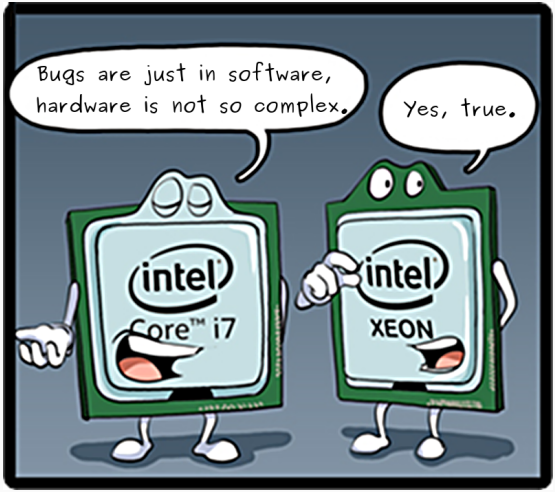
Countermeasures for μ-arch. Attacks

- 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 μ-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



Original image from commitstrip.com

Operating System and CPU Architecture

OS

CPU

Operating System and CPU Architecture

	OS	CPU
High Level	Application	Software

Operating System and CPU Architecture

	OS	CPU
High Level	Application	Software
Interface	Syscall	ISA

Operating System and CPU Architecture

	OS	CPU
High Level	Application	Software
Interface	Syscall	ISA
Low Level	Kernel	Hardware

Operating System and CPU Microarchitecture

OS

CPU

Operating System and CPU Microarchitecture

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

Operating System and CPU Microarchitecture

	OS	CPU
μ-arch. Interface	Non-standard Syscall	ISA Extension
μ-Architecture	Software Caches	Hardware Caches

Michael Schwarz (@misc0110), Erik Kraft

Operating System and CPU Microarchitecture

	OS	CPU
μ-arch. Interface	Non-standard Syscall	ISA Extension
μ-Architecture	Software Caches	Hardware Caches
	Software Prefetcher	Hardware Prefetcher

Operating System and CPU Microarchitecture

	OS	CPU
μ-arch. Interface	Non-standard Syscall	ISA Extension
μ-Architecture	Software Caches	Hardware Caches
	Software Prefetcher	Hardware Prefetcher
	•••	•••

Page Cache

- 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
- State of pages is tracked:

Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
- State of pages is tracked:
  - No write access → clean → no write back

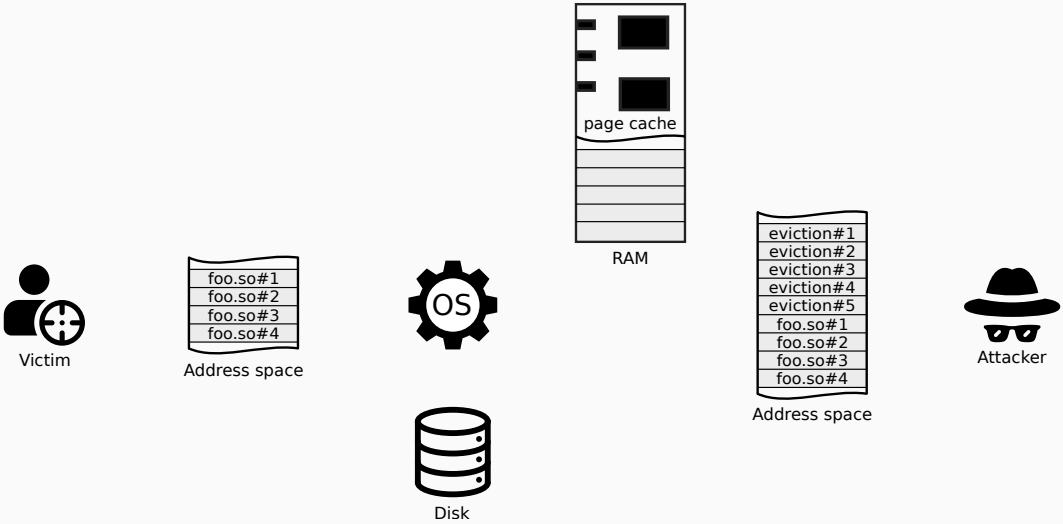
Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
- State of pages is tracked:
  - No write access → clean → no write back
  - Write access → dirty → write back

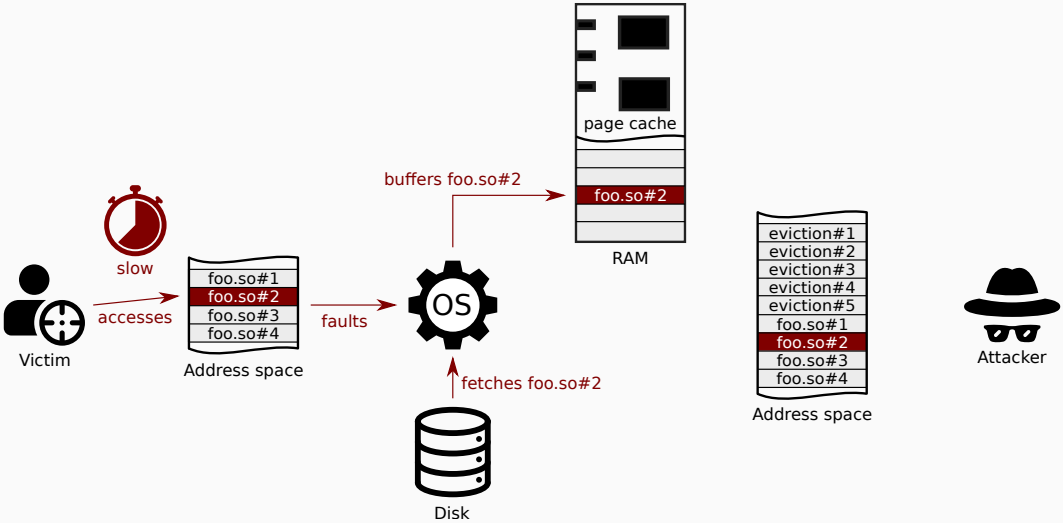
Page Cache

- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
- State of pages is tracked:
  - No write access → clean → no write back
  - Write access → dirty → write back
- Implemented by all major operating systems

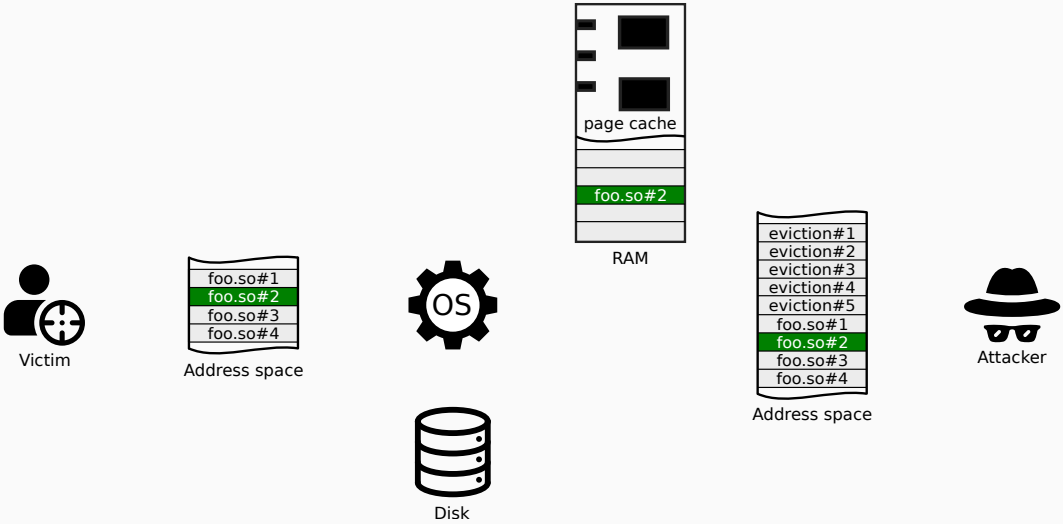
Page Cache Attacks



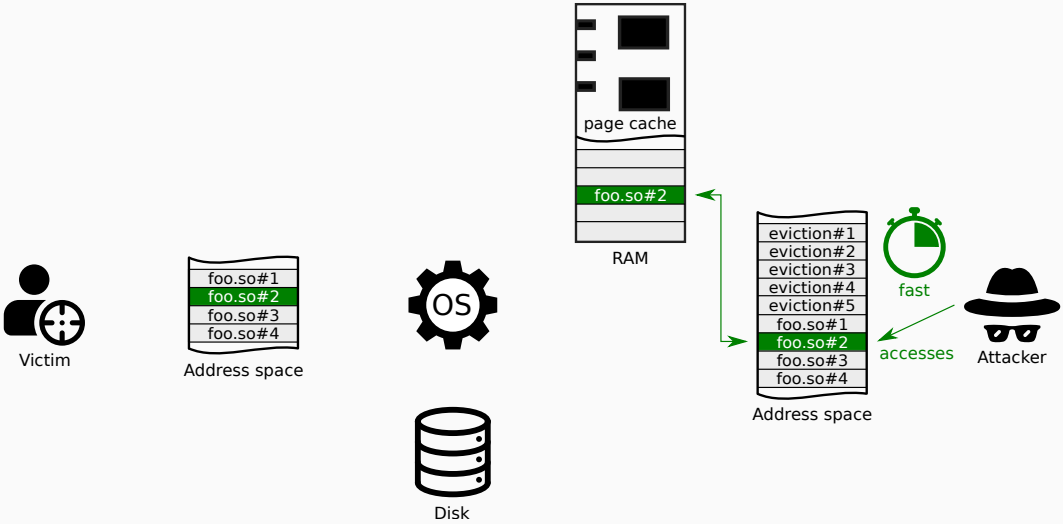
Page Cache Attacks



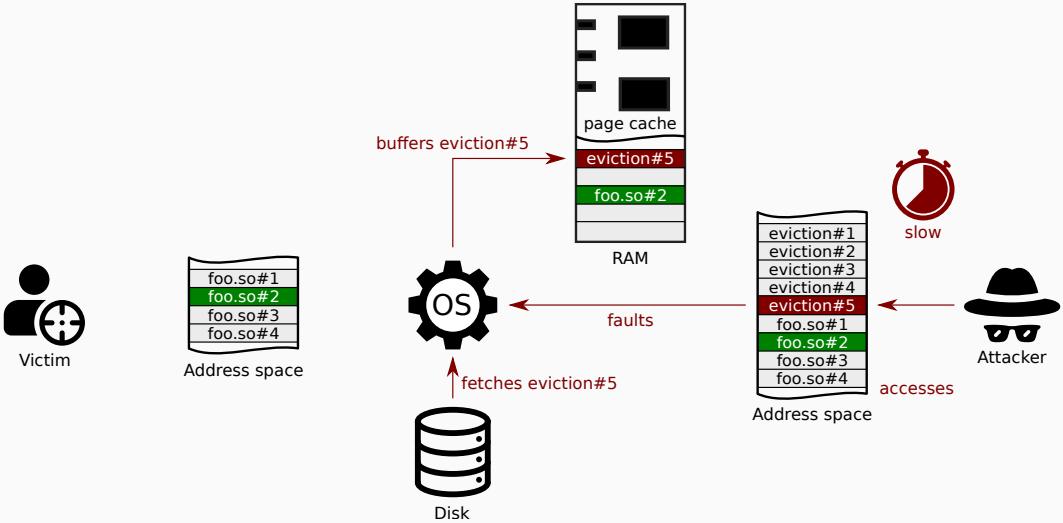
Page Cache Attacks



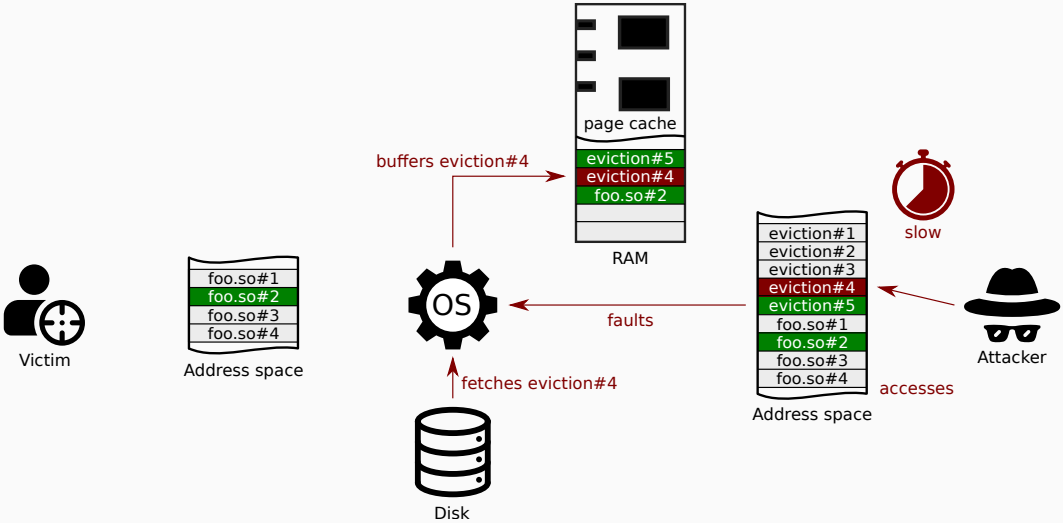
Page Cache Attacks



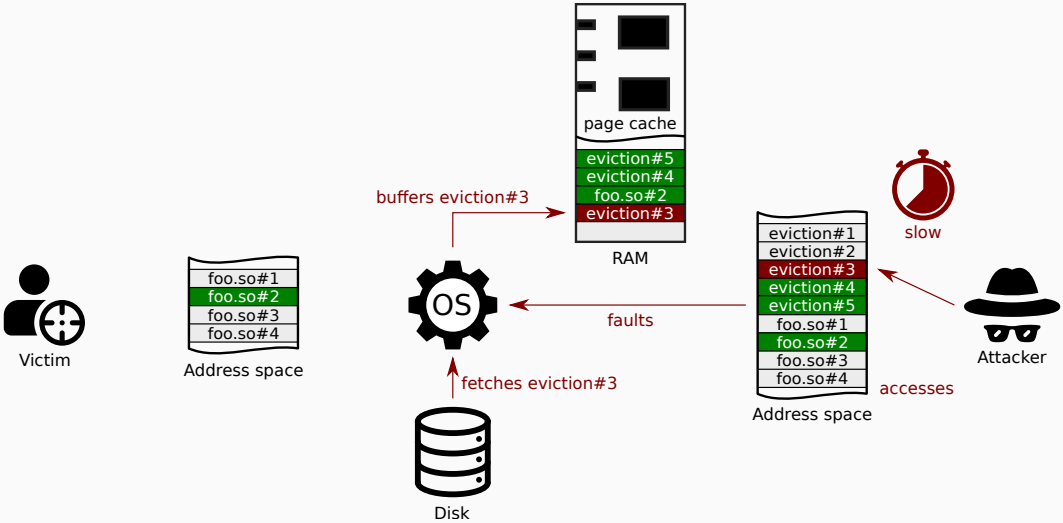
Page Cache Attacks



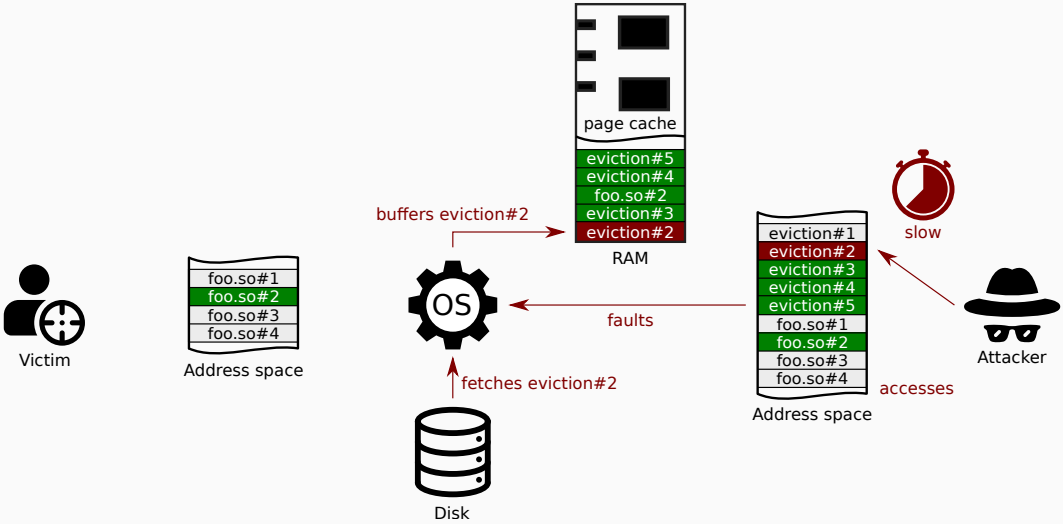
Page Cache Attacks



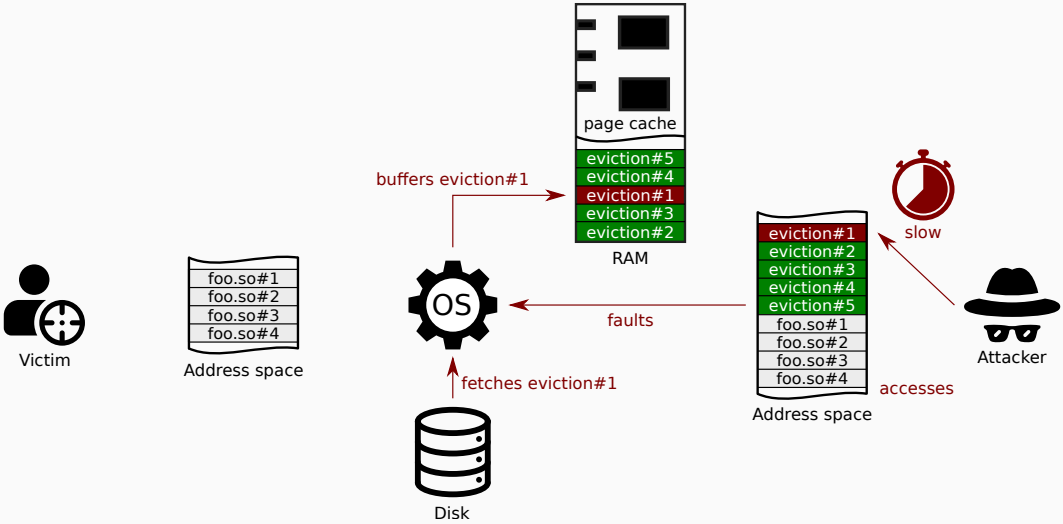
Page Cache Attacks



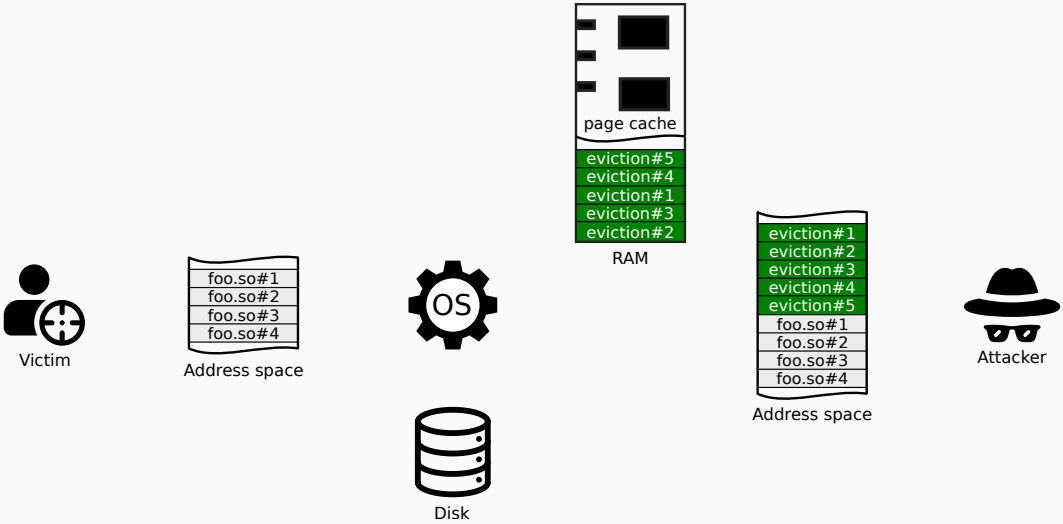
Page Cache Attacks



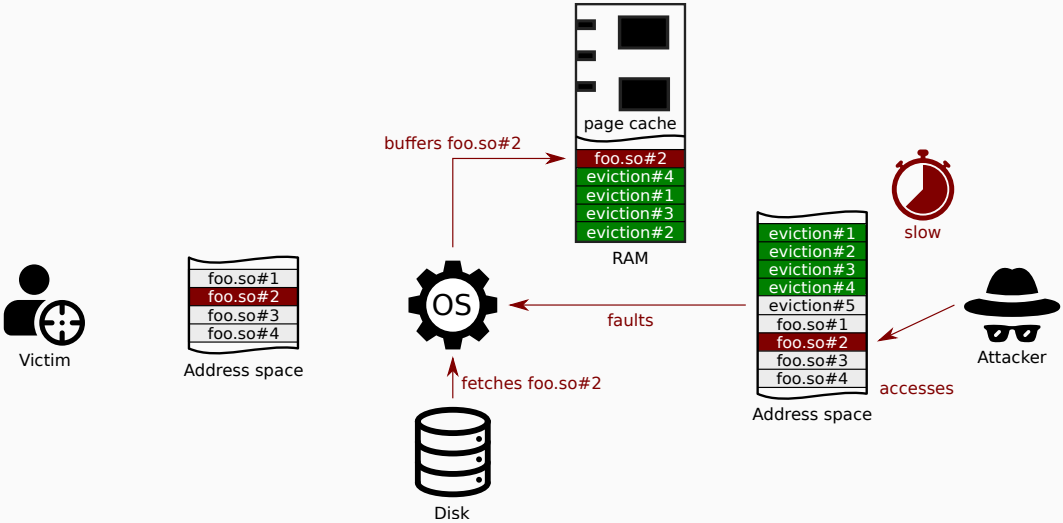
Page Cache Attacks



Page Cache Attacks



Page Cache Attacks



Observe Page Cache State

First idea:

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

Observe Page Cache State

First idea:

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

Observe Page Cache State

First idea:

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

Better:

Observe Page Cache State

First idea:

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

Better:

- Use APIs provided by the operating system
  - `mincore`
  - `QueryWorkingSetEx`
- Non-destructive → higher resolution

Observe Page Cache State

`mincore` (2.04 μs)

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`QueryWorkingSetEx` (465.91 ns)

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`QueryWorkingSetEx` (465.91 ns)

- Takes process handle + virtual memory address, returns struct

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`QueryWorkingSetEx` (465.91 ns)

- Takes process handle + virtual memory address, returns struct
- Exposes attributes of queried page ...

Observe Page Cache State

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`QueryWorkingSetEx` (465.91 ns)

- Takes process handle + virtual memory address, returns struct
- Exposes attributes of queried page ...
- ... presence in working set

Observe Page Cache State

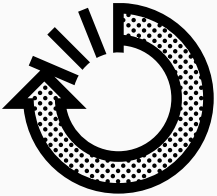
`mincore` (2.04 μs)

- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`QueryWorkingSetEx` (465.91 ns)

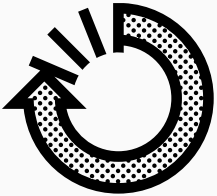
- Takes process handle + virtual memory address, returns struct
- Exposes attributes of queried page ...
- ... presence in working set
- ... number of working sets containing page (`ShareCount`)

Reset Page Cache State



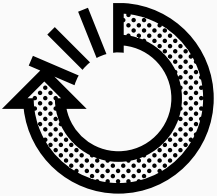
- Necessary for detecting multiple accesses

Reset Page Cache State



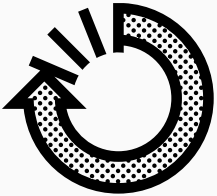
- Necessary for detecting multiple accesses
- Bottleneck of side channel

Reset Page Cache State



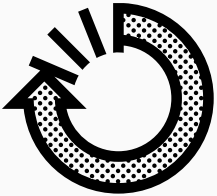
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on page cache implementation

Reset Page Cache State



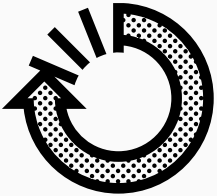
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on page cache implementation
  - Differences in page replacement

Reset Page Cache State



- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on page cache implementation
  - Differences in page replacement
  - Global CLOCK-Pro like algorithm

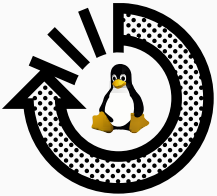
Reset Page Cache State



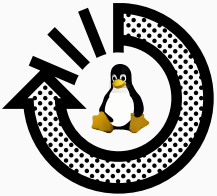
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on page cache implementation
  - Differences in page replacement
  - Global CLOCK-Pro like algorithm
  - Per-process working sets with Aging algorithm

Eviction Linux

- Access pages until target page is replaced

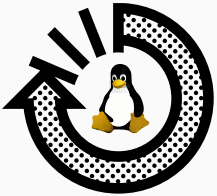


Eviction Linux



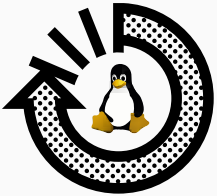
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file

Eviction Linux



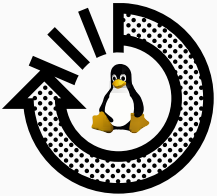
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache

Eviction Linux



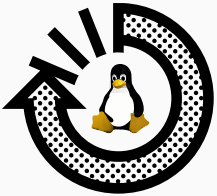
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction

Eviction Linux



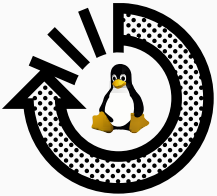
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system

Eviction Linux



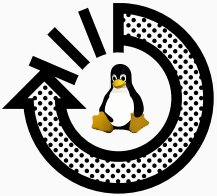
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system
- Optimisation 2: Fill memory with anonymous dirty pages

Eviction Linux



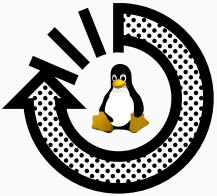
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system
- Optimisation 2: Fill memory with anonymous dirty pages
  - Very effective if swapping disabled

Eviction Linux



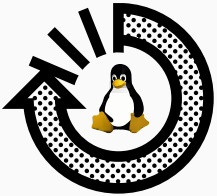
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system
- Optimisation 2: Fill memory with anonymous dirty pages
  - Very effective if swapping disabled
  - Decreases average eviction set size

Eviction Linux



- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system
- Optimisation 2: Fill memory with anonymous dirty pages
  - Very effective if swapping disabled
  - Decreases average eviction set size
  - Significant influence on eviction time and stability

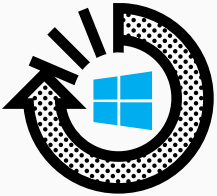
Eviction Linux



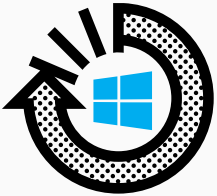
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- Optimisation 1: Add pages already in page cache
  - Target page more likely selected for eviction
  - Higher responsiveness of system
- Optimisation 2: Fill memory with anonymous dirty pages
  - Very effective if swapping disabled
  - Decreases average eviction set size
  - Significant influence on eviction time and stability
- Average run time down to 149 ms depending on optimisations

Eviction Windows

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

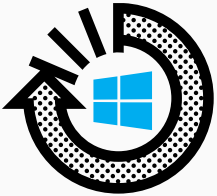


Eviction Windows



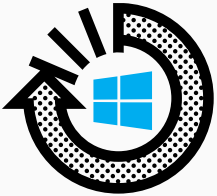
- Page cache eviction ↔ target page drops out of all working sets
  - Previous approach slow...

Eviction Windows



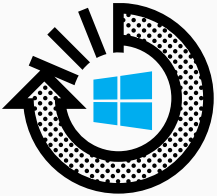
- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation

Eviction Windows



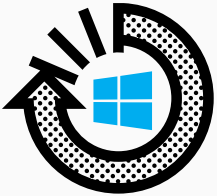
- Page cache eviction ↔ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size

Eviction Windows



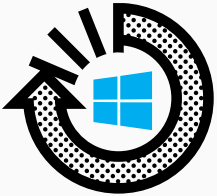
- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction ($<2\,s$)

Eviction Windows



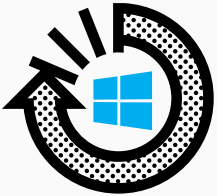
- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction (<2 s)
  - Eviction from own ws $\rightarrow$ `VirtualUnlock` (17.69 μs)
  - Eviction other ws $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)

Eviction Windows



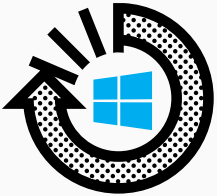
- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction ($<2\,\mathrm{s}$)
  - Eviction from own ws $\rightarrow$ `VirtualUnlock` ($17.69\,\mu\mathrm{s}$)
  - Eviction other ws $\rightarrow$ `SetProcessWorkingSetSize` ($4.48\,\mathrm{ms}$)
    - for processes with same integrity level as attacker

Eviction Windows



- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction (<2 s)
  - Eviction from own ws $\rightarrow$ `VirtualUnlock` (17.69 μs)
  - Eviction other ws $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)
    - for processes with same integrity level as attacker
  - Observe via `QueryWorkingSetEx`

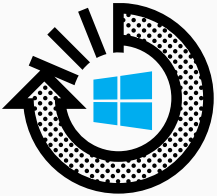
Eviction Windows



- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction (<2 s)
  - Eviction from own ws $\rightarrow$ `VirtualUnlock` (17.69 μs)
  - Eviction other ws $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)
    - for processes with same integrity level as attacker
  - Observe via `QueryWorkingSetEx`
    - on own process (`ShareCount`)

Michael Schwarz (@misc0110), Erik Kraft

Eviction Windows

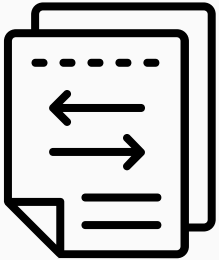


- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Different approach: Use working-set eviction + observation
  - Limited maximum ws size
  - Apply memory pressure $\rightarrow$ self-eviction (<2 s)
  - Eviction from own ws $\rightarrow$ `VirtualUnlock` (17.69 μs)
  - Eviction other ws $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)
    - for processes with same integrity level as attacker
  - Observe via `QueryWorkingSetEx`
    - on own process (`ShareCount`)
    - on other process (same integrity level as attacker)

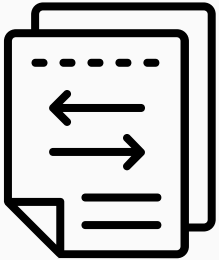
Michael Schwarz (@misc0110), Erik Kraft

Covert Channel

- Shared file as information carrier

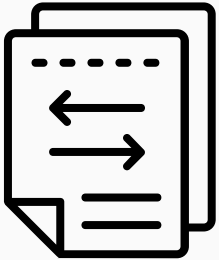


Covert Channel



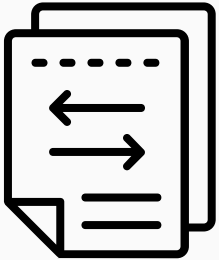
- Shared file as information carrier
- File page presence in page cache ↔ message bits

Covert Channel



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

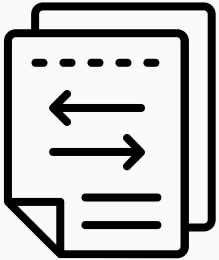
Covert Channel



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

Different implementation approaches:

Covert Channel

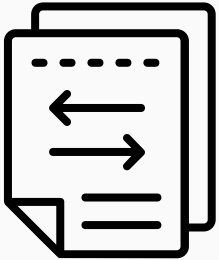


- Shared file as information carrier
- File page presence in page cache ↔ message bits
- Additional file pages for transmission control

Different implementation approaches:

OS	Eviction	Observation	Speed
Linux	like side channel	`mincore`	20.20 kB/s
	`madvise` `posix_fadvise`	`mincore`	81.16 kB/s
Windows	process WS `VirtualUnlock`	`QueryWorkingSetEx` (`ShareCount`)	100.11 kB/s

Covert Channel



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

Different implementation approaches:

OS	Eviction	Observation	Speed
Linux	like side channel	`mincore`	20.20 kB/s
	`madvise` `posix_fadvise`	`mincore`	81.16 kB/s
Windows	process WS `VirtualUnlock`	`QueryWorkingSetEx` (`ShareCount`)	100.11 kB/s

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

PHP RNG Attack

- Targets seeding of PHP PRNG

PHP RNG Attack

- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks

PHP RNG Attack

- 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 :(

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

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

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 RNG Attack

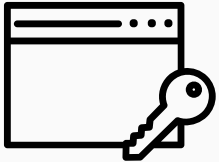
- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to detect `microtime` call
  - Seed recoverable
- `zif_microtime` on page 781 of `php-fpm` executable
  - `PHP 7.3.5`, 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: ± 1 ms

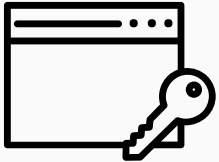
Live Demo

UI Redressing Attack



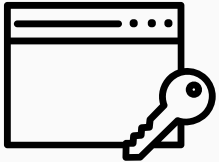
- Detect opening of interesting window

UI Redressing Attack



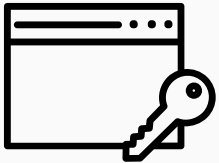
- Detect opening of interesting window
  - e.g. authentication windows

UI Redressing Attack



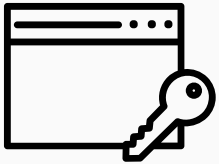
- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake

UI Redressing Attack



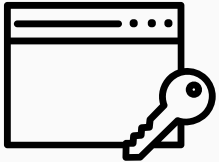
- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger

UI Redressing Attack



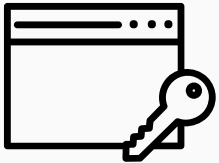
- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger
- Provides low latency → hardly noticeable

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`

UI Redressing Attack

Impact

- Identified as CVE-2019-5489

Impact

- Identified as CVE-2019-5489
  - CVSS v3.0: 5.5 MEDIUM

Impact

- Identified as CVE-2019-5489
  - CVSS v3.0: 5.5 MEDIUM
- Findings addressed by Linux and Windows

Impact

- Identified as CVE-2019-5489
  - CVSS v3.0: 5.5 MEDIUM
- Findings addressed by Linux and Windows
  - Countermeasures developed

Countermeasures

- Increase observation effort

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort
  - Use a local page replacement approach

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort
  - Use a local page replacement approach
  - e.g. per-process working sets

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort
  - Use a local page replacement approach
  - e.g. per-process working sets
  - Page evicted only if in no working set

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort
  - Use a local page replacement approach
  - e.g. per-process working sets
  - Page evicted only if in no working set
  - No direct influence on replacement choices between processes

Countermeasures

- Increase observation effort
  - Higher privileges for leaking APIs
  - Access times still can be used ...
  - ... but they destroy state → additional evictions
  - Eviction is the bottleneck → lower resolution
- Increase eviction effort
  - Use a local page replacement approach
  - e.g. per-process working sets
  - Page evicted only if in no working set
  - No direct influence on replacement choices between processes
  - Harder to evict pages → lower frequency

Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes

Countermeasures - Windows

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

Countermeasures - Windows

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

Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` removed
  - No indirect querying of working set state

Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` removed
  - No indirect querying of working set state
- Non-destructive probing no longer possible

Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` removed
  - No indirect querying of working set state
- Non-destructive probing no longer possible
  Working set probing no longer possible → weaker attack

Countermeasures - Windows

- Higher privilege for `QueryWorkingSetEx` on other processes
  - No direct querying of working set state
- `ShareCount` removed
  - No indirect querying of working set state
- Non-destructive probing no longer possible
  Working set probing no longer possible → 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` removed
  - No indirect querying of working set state
- Non-destructive probing no longer possible
  Working set probing no longer possible → weaker attack
  - If `QueryWorkingSetEx` only possible leakage source
  - Page-cache eviction already harder than on Linux

Countermeasures - Linux

- `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

Countermeasures - Linux

- `mincore` only reveals information for writeable file pages
- Read-only files excluded → shared libraries, executables
- Newest patch, not in mainline yet

Countermeasures - Linux

- `mincore` only reveals information for writeable file pages
- Read-only files excluded → shared libraries, executables
- Newest patch, not in mainline yet
- Non-destructive probing no longer possible?

Countermeasures - Linux

- `mincore` only reveals information for writeable file pages
- Read-only files excluded → shared libraries, executables
- Newest patch, not in mainline yet
- 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
- Newest patch, not in mainline yet
- Non-destructive probing no longer possible?
- No, `preadv2` with `RWF_NOWAIT` leaks same information
  - Countermeasure currently under development

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

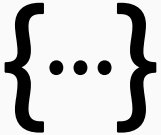
The Future

- We won't get rid of side channels
- More optimizations $\rightarrow$ more side channels

The Future

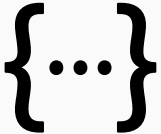
- We won't get rid of side channels
- More optimizations → more side channels
- More attacks on the "OS microarchitecture"

Take Aways



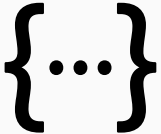
- Abstraction leads to side channels

Take Aways



- Abstraction leads to side channels
- Software-cache attacks are similar to hardware-cache attacks

Take Aways



- Abstraction leads to side channels
- Software-cache attacks are similar to hardware-cache attacks
- Finding countermeasures is difficult

RuhrSec

Are Microarchitectural Attacks still possible on Flawless Hardware?

Michael Schwarz
Erik Kraft

Acknowledgements

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.