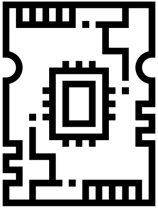
# Page Cache Attacks

## Microarchitectural Attacks on Flawless Hardware

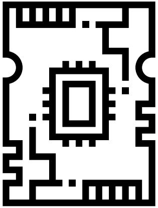
Daniel Gruss, Trishita Tiwari, Michael Schwarz, Erik Kraft

# Microarchitectural Components



- Modern CPUs contain multiple microarchitectural elements

# Microarchitectural Components



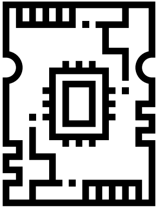
- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

...

# Microarchitectural Components



- Modern CPUs contain multiple microarchitectural elements

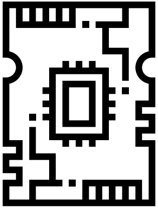
Caches and buffers

Predictors

...

- Transparent for the programmer

# Microarchitectural Components



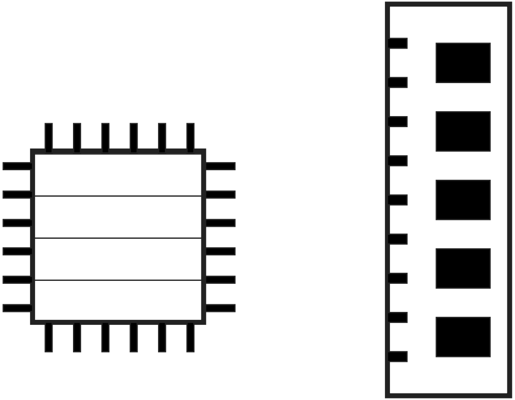
- Modern CPUs contain multiple microarchitectural elements

Caches and buffers

Predictors

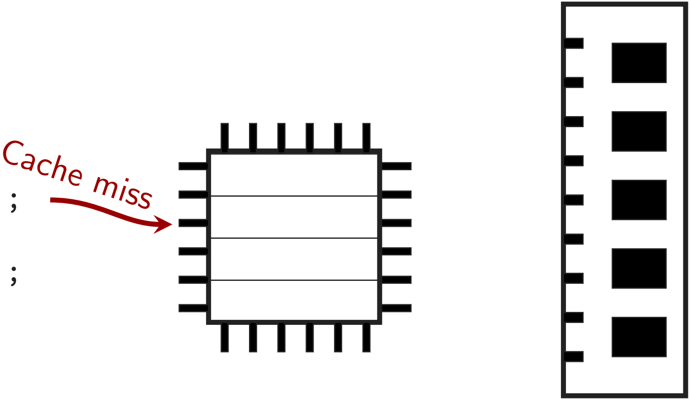
- Transparent for the programmer
- Timing optimizations → side-channel leakage

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



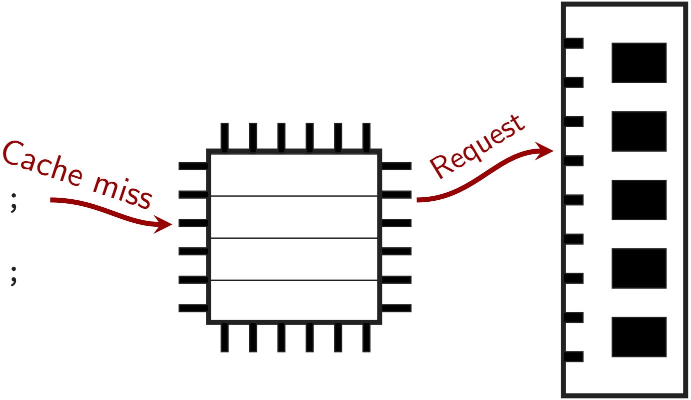
# CPU Cache

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

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



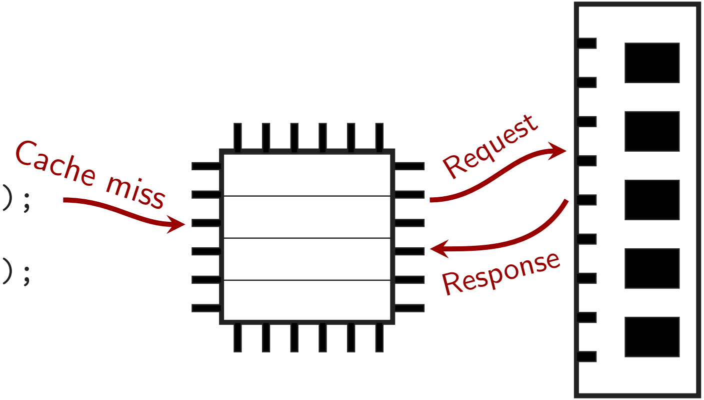
# CPU Cache

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

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

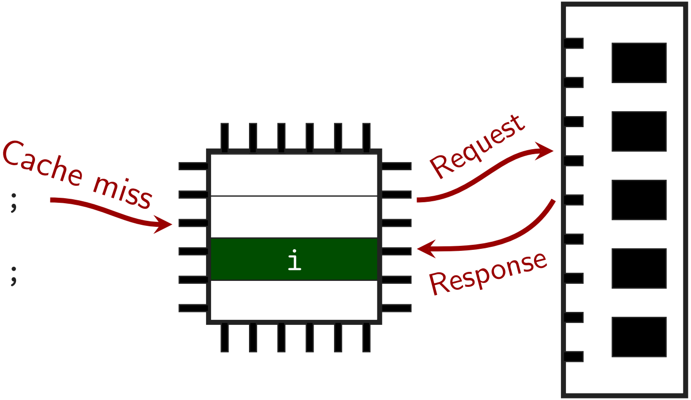


# CPU Cache

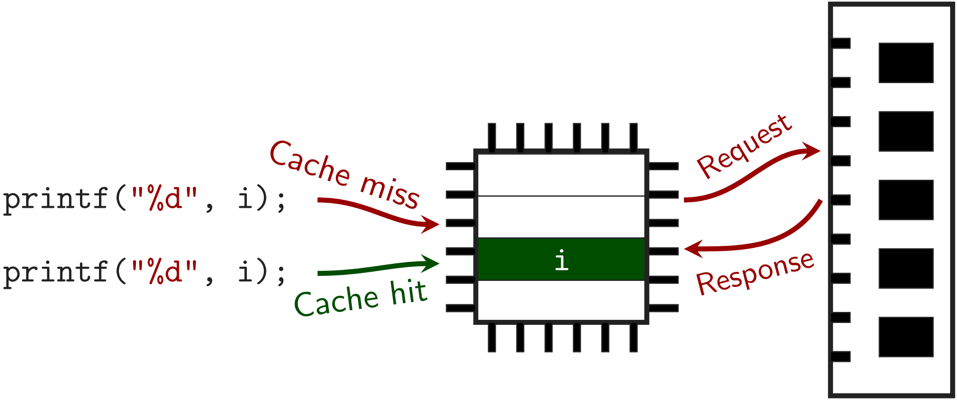
```
printf("%d", i);

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



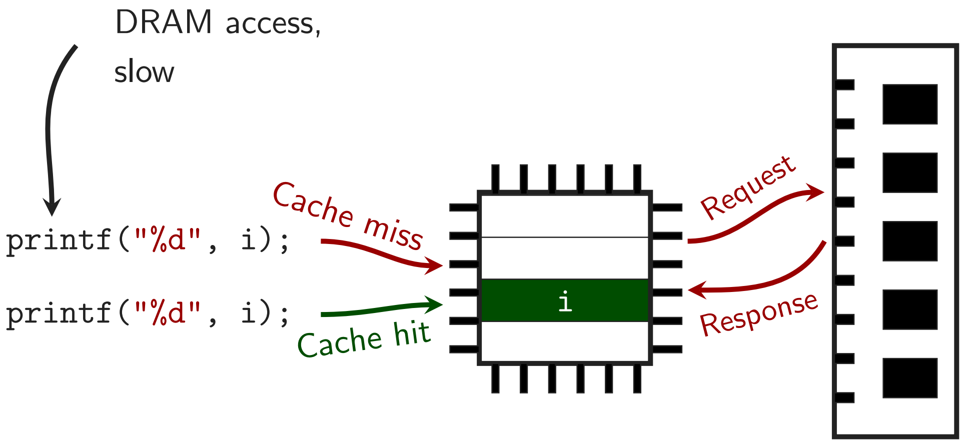
```
printf("%d", i);

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



# CPU Cache



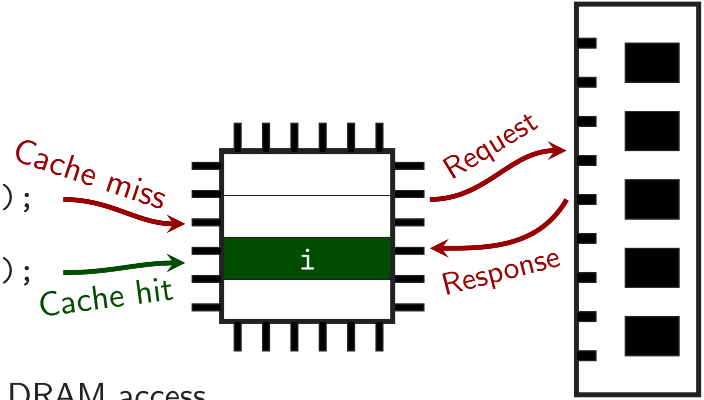
# CPU Cache


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

# CPU Cache

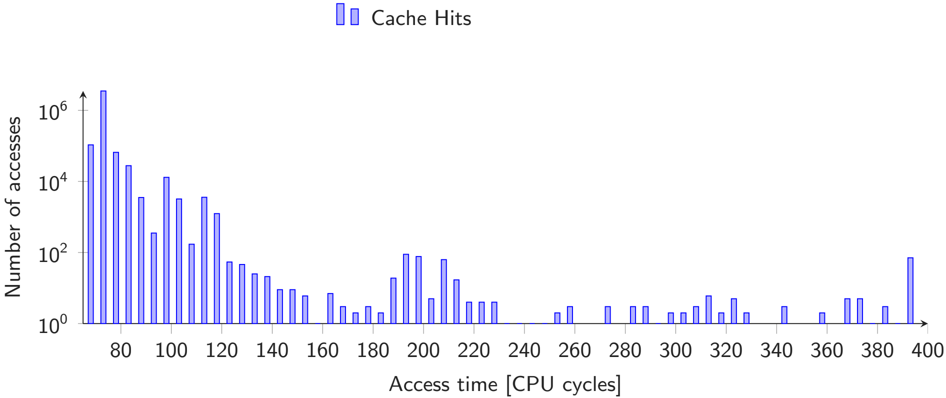
DRAM access,
slow

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

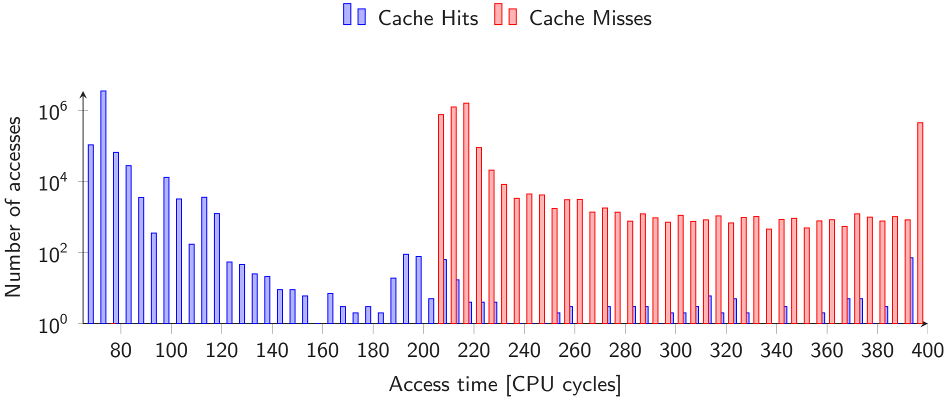



No DRAM access,
much faster

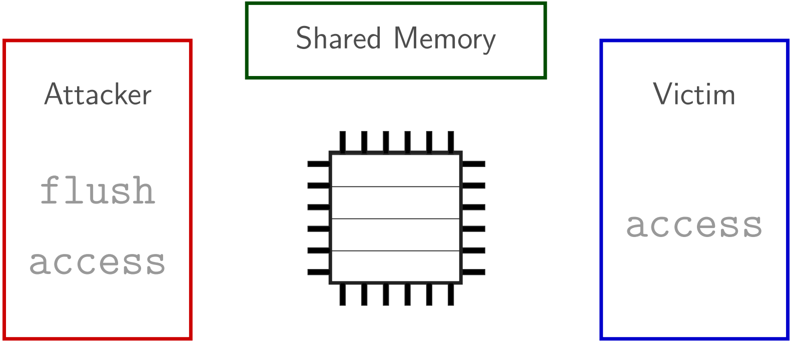
# Caching speeds up Memory Accesses

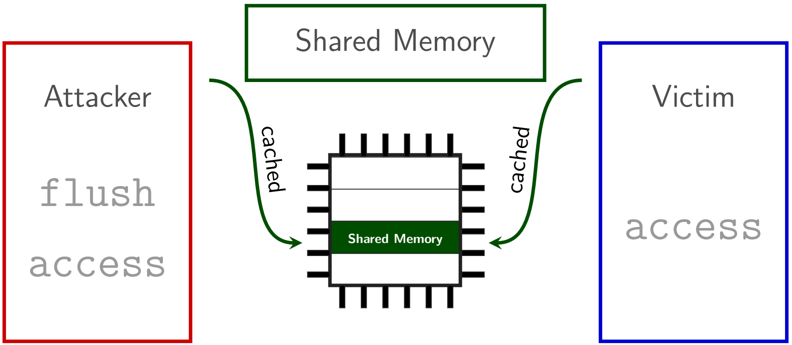


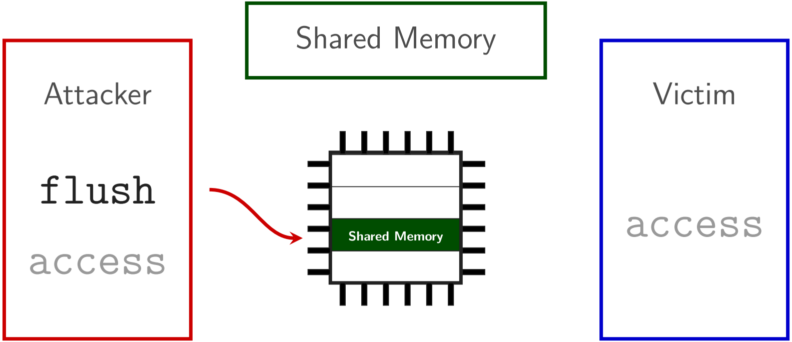
# Caching speeds up Memory Accesses

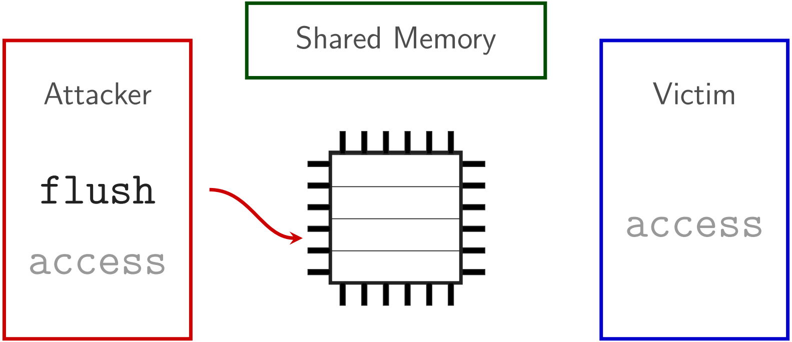


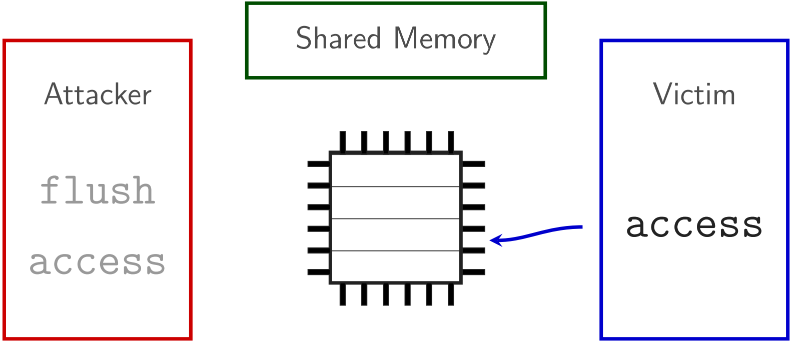
# Flush+Reload

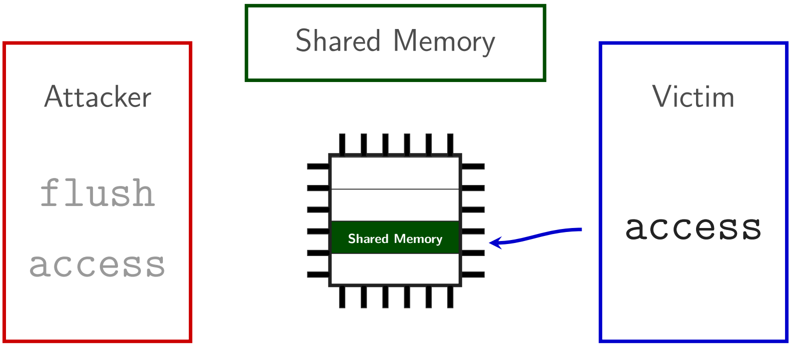


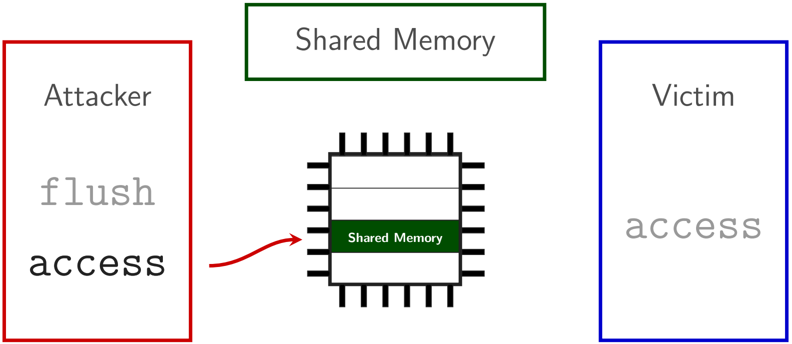

Shared Memory
Attacker
flush
access
cached
Shared Memory
cached
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Victim
access

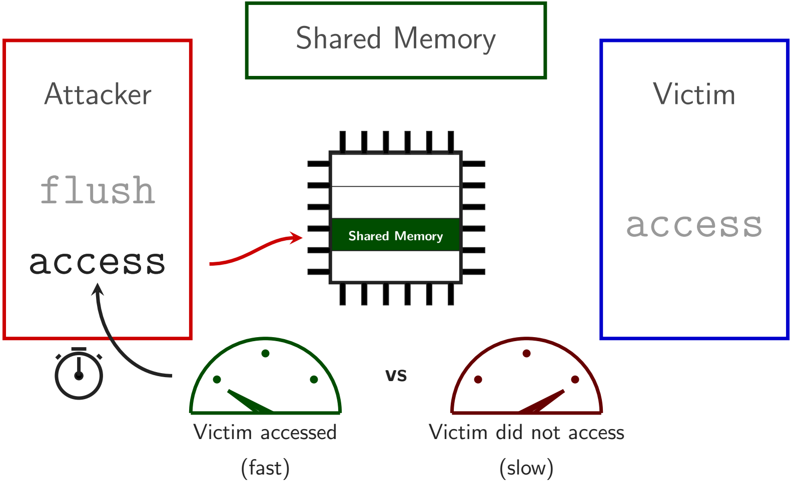

Attacker
flush
access
Shared Memory
Victim
access


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access

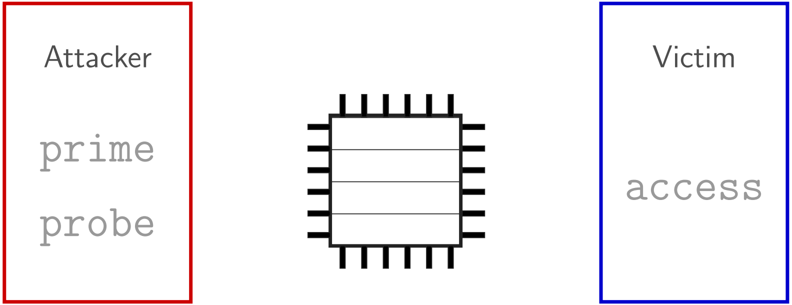


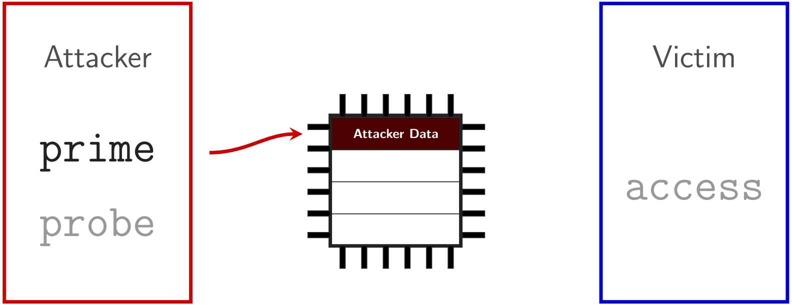


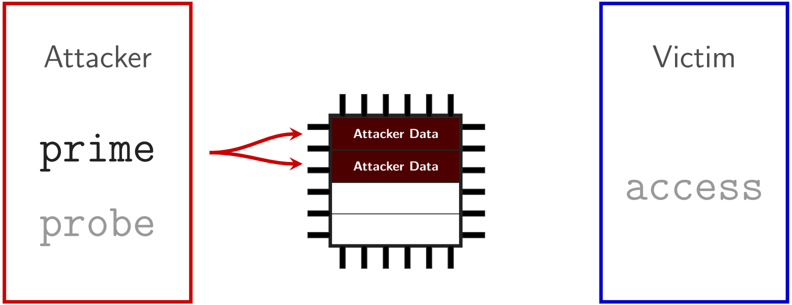
Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)


Shared Memory
Attacker
flush
access
Shared Memory
Victim
access
vs
Victim accessed
(fast)
Victim did not access
(slow)

# Prime+Probe

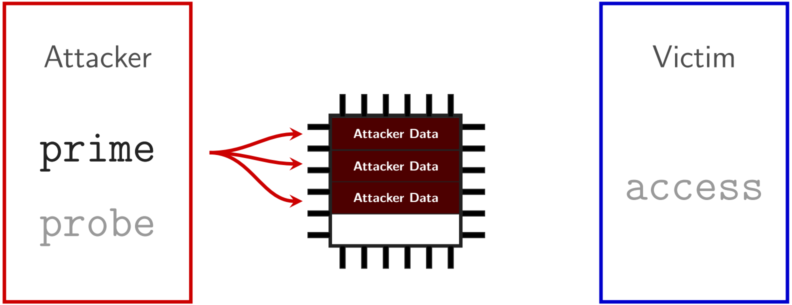



Attacker
prime
probe
Attacker Data
Victim
access

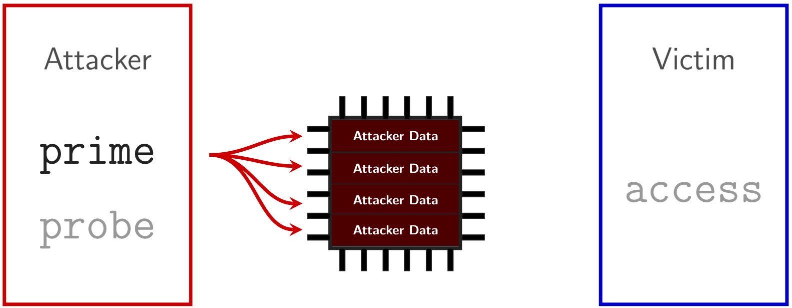


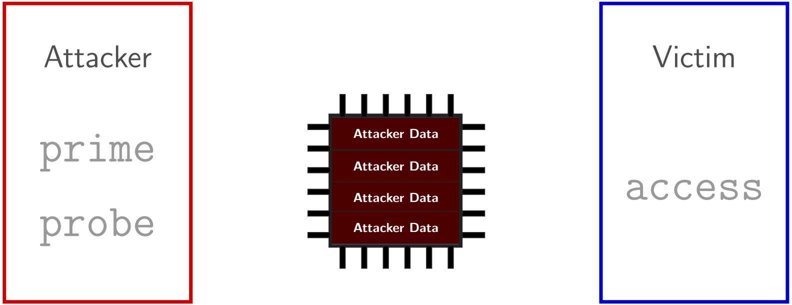


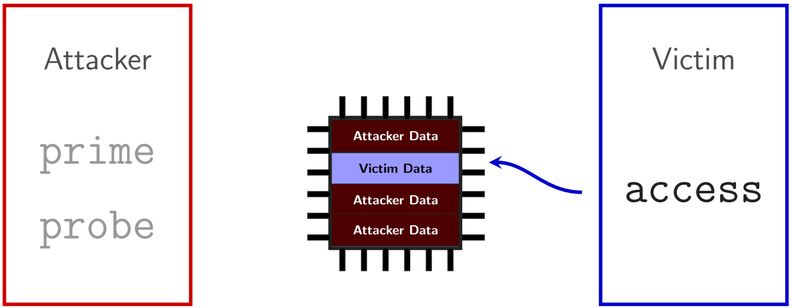
Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)

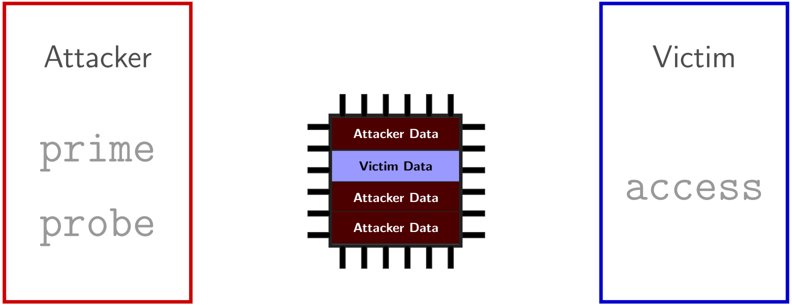

Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Victim
access

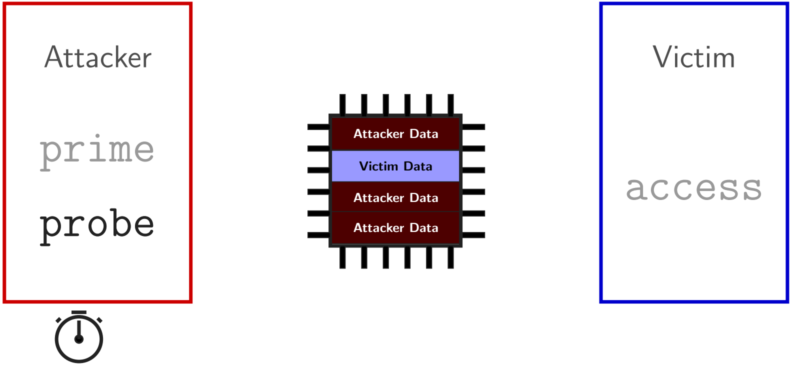
# Prime+Probe

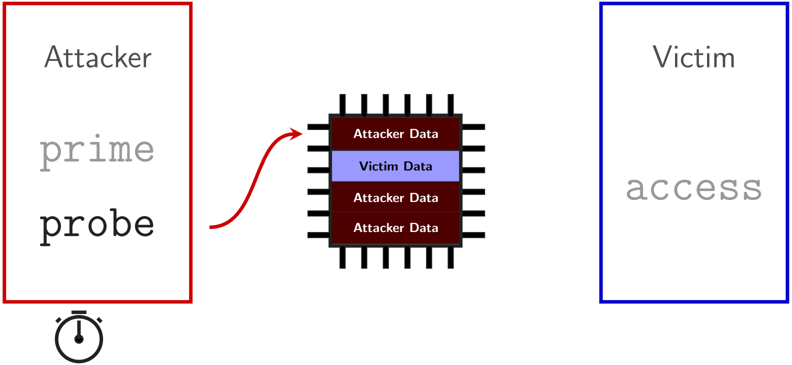

Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Attacker Data
Victim
access

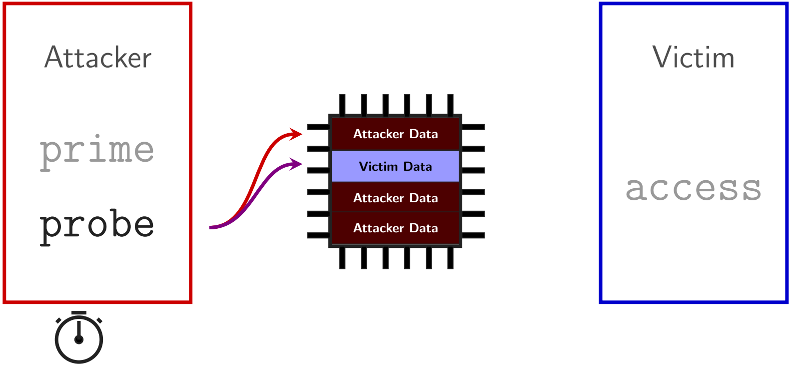

Attacker
prime
probe
Attacker Data
Attacker Data
Attacker Data
Attacker Data
Victim
access


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

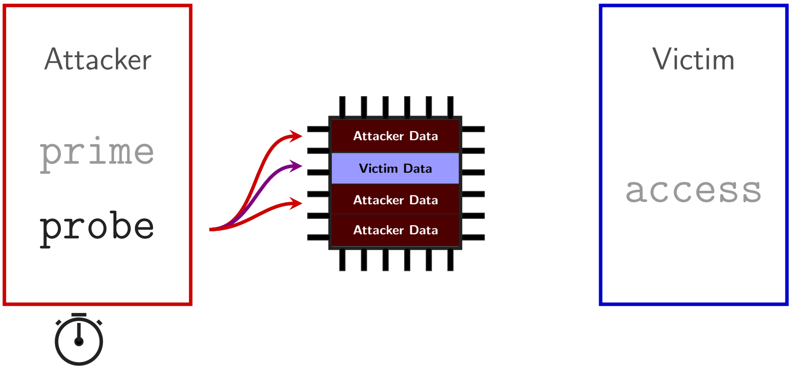

Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access


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

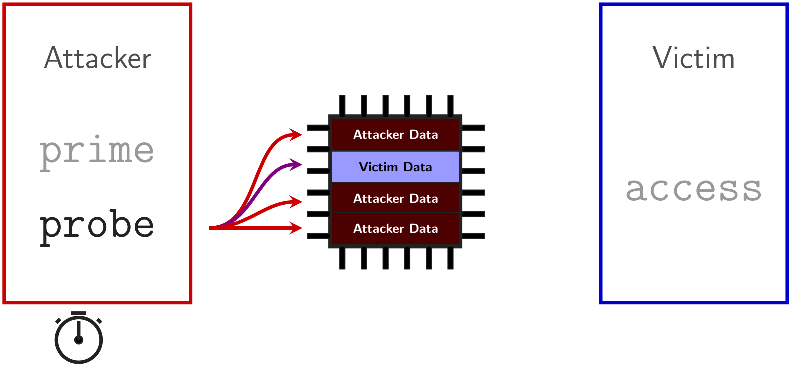

Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access


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

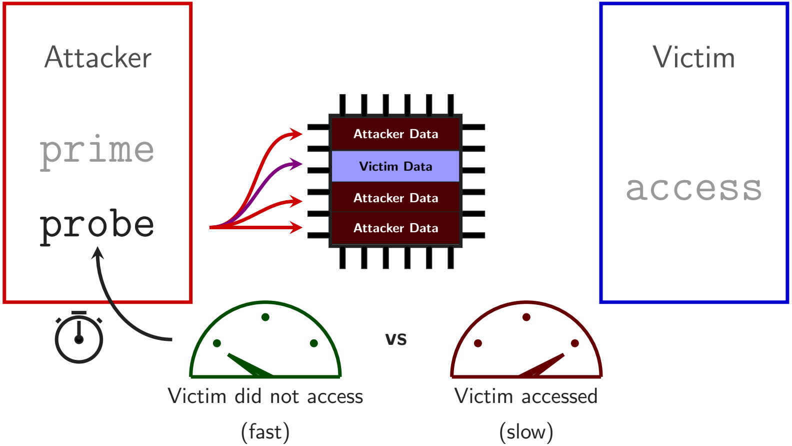
# Prime+Probe


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

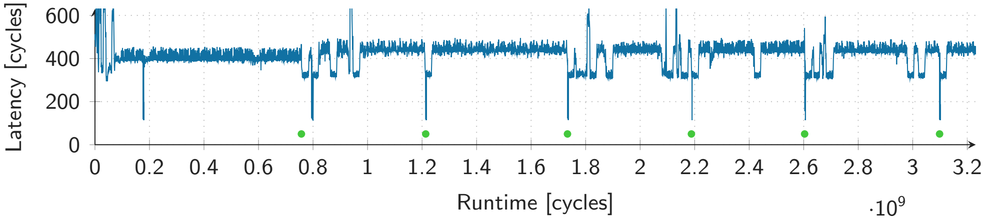
# Prime+Probe


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

# Prime+Probe

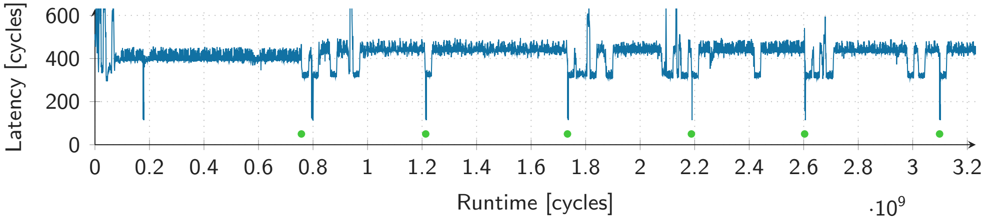

Attacker
prime
probe
Attacker Data
Victim Data
Attacker Data
Attacker Data
Victim
access
vs
Victim did not access
(fast)
Victim accessed
(slow)

# Flush+Reload on Keystrokes




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

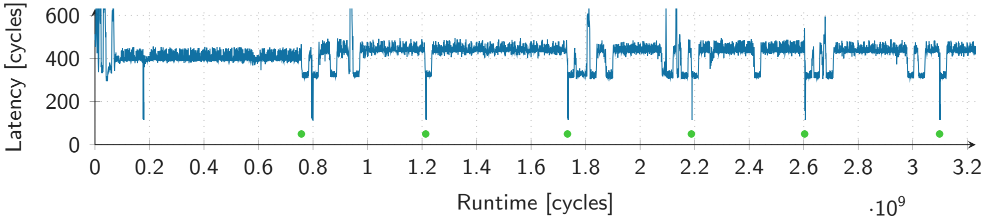
# Flush+Reload on Keystrokes




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

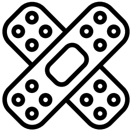

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)

# Flush+Reload on Keystrokes



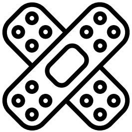

- Key presses trigger code execution in shared library (e.g., `libgdk`)
- Flush+Reload does not reveal actual key, only time difference between keys
- → Recover text with machine learning

# Meltdown, Foreshadow, ZombieLoad, Spectre

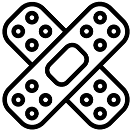


- Deeply rooted in hardware $\rightarrow$ no real fixes

# Countermeasures for $\mu$-arch. Attacks

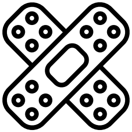


- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder



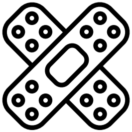
- Deeply rooted in hardware $\rightarrow$ no real fixes
- More isolation $\rightarrow$ make exploitation harder
- Attacks on design difficult to fix

# Countermeasures for $\mu$-arch. Attacks



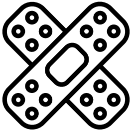
- Deeply rooted in hardware → no real fixes
- More isolation → make exploitation harder
- Attacks on design difficult to fix
  - Caches → we want timing differences

# Countermeasures for $\mu$-arch. Attacks

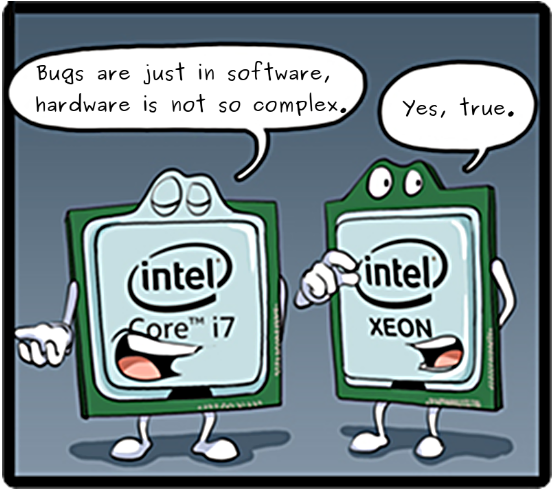


- Deeply rooted in hardware → no real fixes
- More isolation → make exploitation harder
- Attacks on design difficult to fix
    - Caches → we want timing differences
    - Prediction → we don't want stalls

# Countermeasures for $\mu$-arch. Attacks



- Deeply rooted in hardware → no real fixes
- More isolation → make exploitation harder
- Attacks on design difficult to fix
  - Caches → we want timing differences
  - Prediction → we don't want stalls
- So far: fixing symptoms




Original image from commitstrip.com

Thought experiment: what if there were no hardware side channels?

# Operating System and CPU Microarchitecture

OS

CPU

# Operating System and CPU Microarchitecture

| | OS | CPU |
|---|---|---|
| **$\mu$-arch. Interface** | Non-standard Syscall | x86<br>ISA Extension |

# Operating System and CPU Microarchitecture

| | OS | CPU |
|---|---|---|
| **$\mu$-arch. Interface** | Non-standard Syscall | x86 ISA Extension |
| **$\mu$-Architecture** | Software Caches | Hardware Caches |

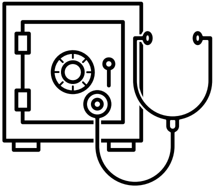
# Operating System and CPU Microarchitecture

| | OS | CPU |
|---|---|---|
| $\mu$-**arch. Interface** | Non-standard Syscall | x86 ISA Extension |
| $\mu$-**Architecture** | Software Caches | Hardware Caches |
| | Software Prefetcher | Hardware Prefetcher |

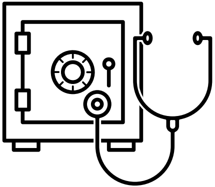

Daniel Gruss (@lavados), Trishita Tiwari (@fork_while_1), Michael Schwarz (@misc0110), Erik Kraft (@ekraft95)

# Operating System and CPU Microarchitecture

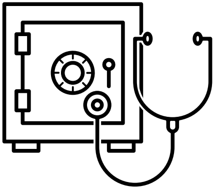
| | OS | CPU |
|---|---|---|
| $\mu$-**arch. Interface** | Non-standard Syscall | x86 ISA Extension |
| $\mu$-**Architecture** | Software Caches | Hardware Caches |
| | Software Prefetcher | Hardware Prefetcher |
| | ... | ... |



- Hardware $\rightarrow$ Software?

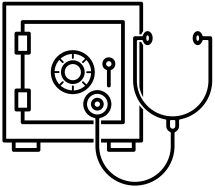


- Hardware → Software?
- Hardware-Agnostic Side Channel through the OS Page Cache



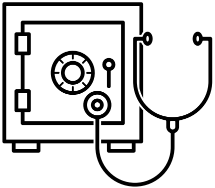
- Hardware → Software?
- Hardware-Agnostic Side Channel through the OS Page Cache
- Temporal resolution:
  - 2 µs ($\leq$ 6.7 measurements per second)
  - 466 ns ($\leq$ 223 measurements per second)

# OS Side Channel

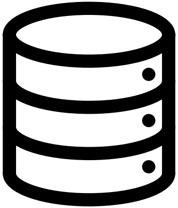


- Hardware $\rightarrow$ Software?
- Hardware-Agnostic Side Channel through the OS Page Cache
- Temporal resolution:
  - 2 µs ($\leq$ 6.7 measurements per second)
  - 466 ns ($\leq$ 223 measurements per second)
- Spatial resolution of 4 KiB

# OS Side Channel

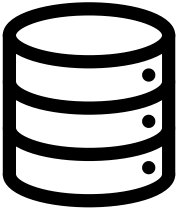


- Hardware → Software?
- Hardware-Agnostic Side Channel through the OS Page Cache
- Temporal resolution:
  - 2 µs ($\leq$ 6.7 measurements per second)
  - 466 ns ($\leq$ 223 measurements per second)
- Spatial resolution of 4 KiB
- Various attacks: PHP RNG, UI-Redress, Windows ASLR, Keystroke Timings,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

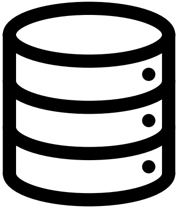


- Managed by operating system

# Page Cache

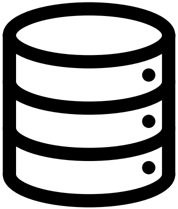


- Managed by operating system
- Buffers file pages in RAM for faster accesses



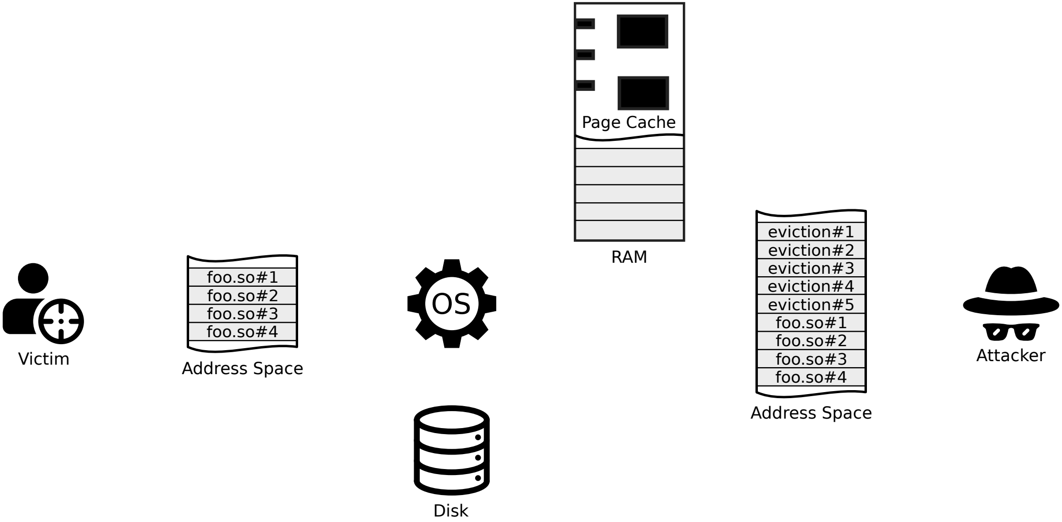
- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache

# Page Cache

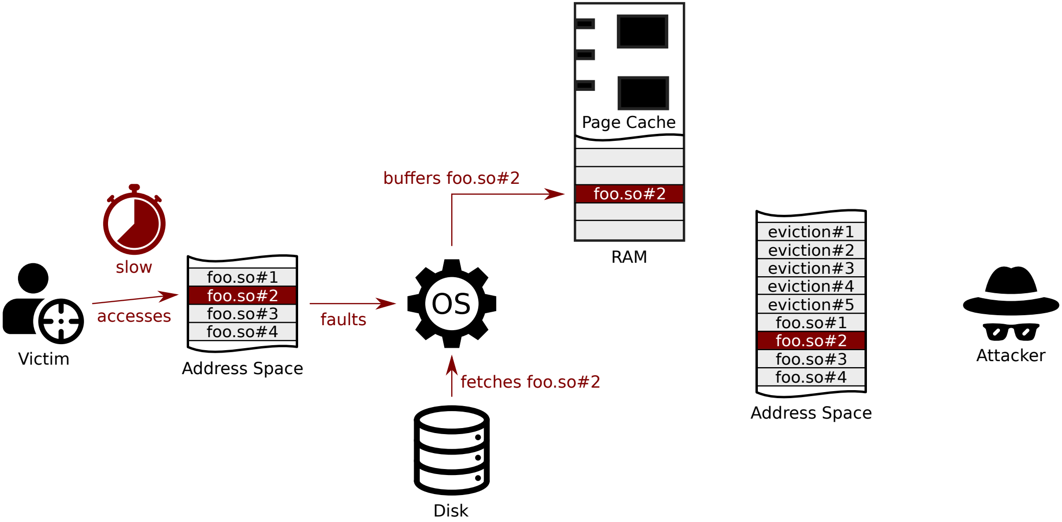


- Managed by operating system
- Buffers file pages in RAM for faster accesses
- Ideally all file pages in page cache
- Implemented by all major operating systems

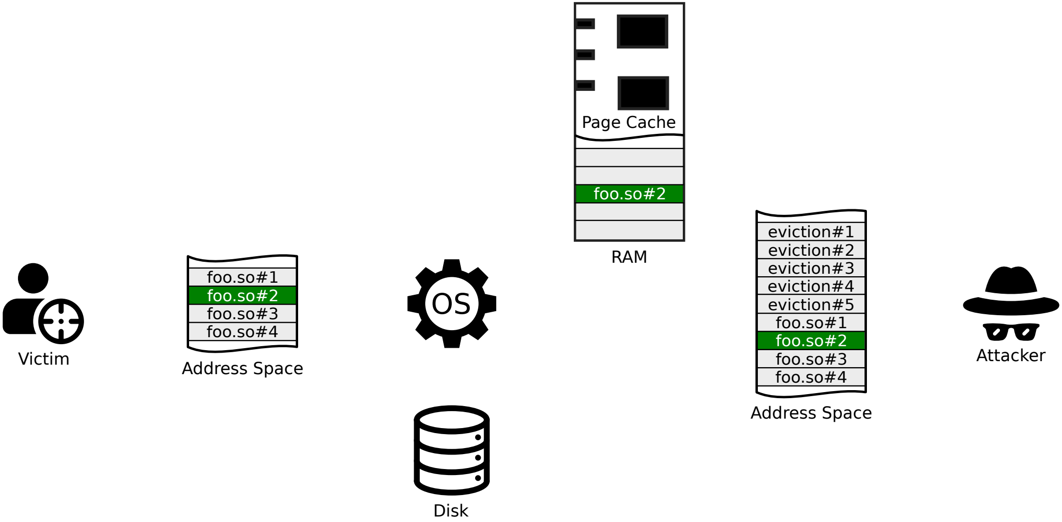
# Page Cache Attacks



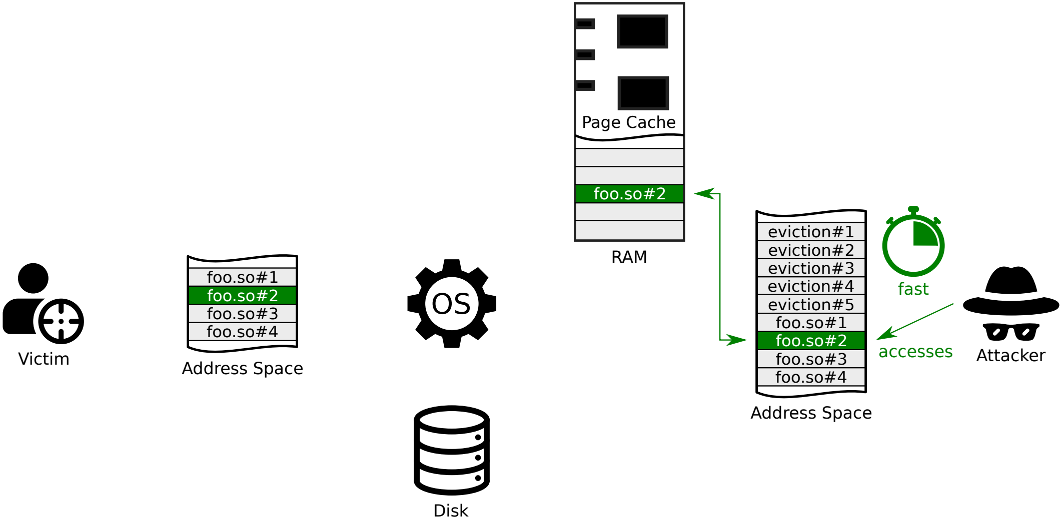
# Page Cache Attacks



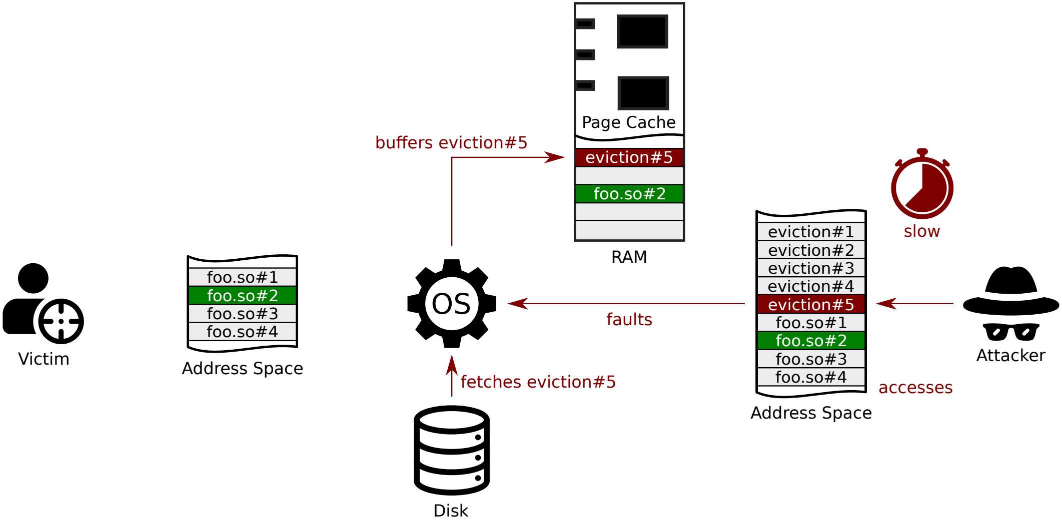
# Page Cache Attacks



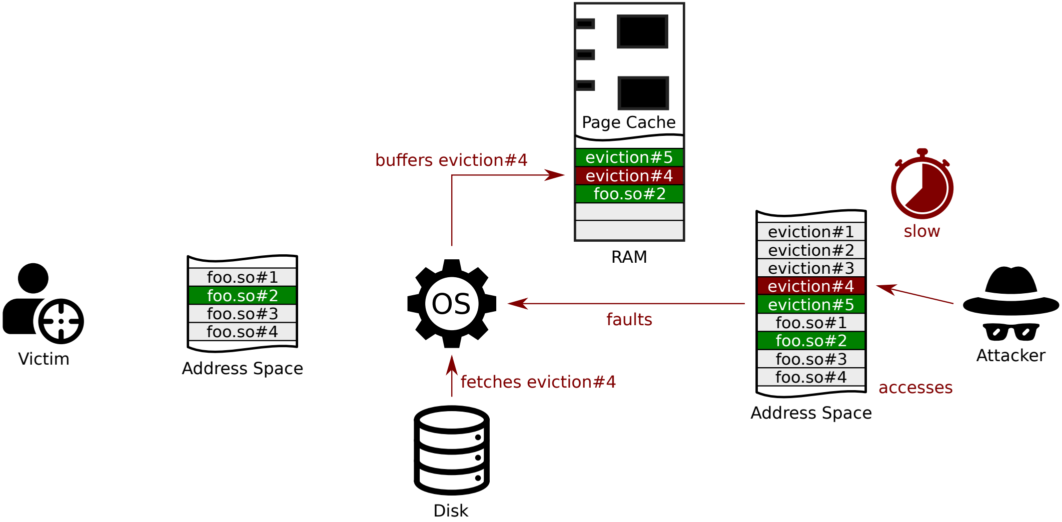
# Page Cache Attacks



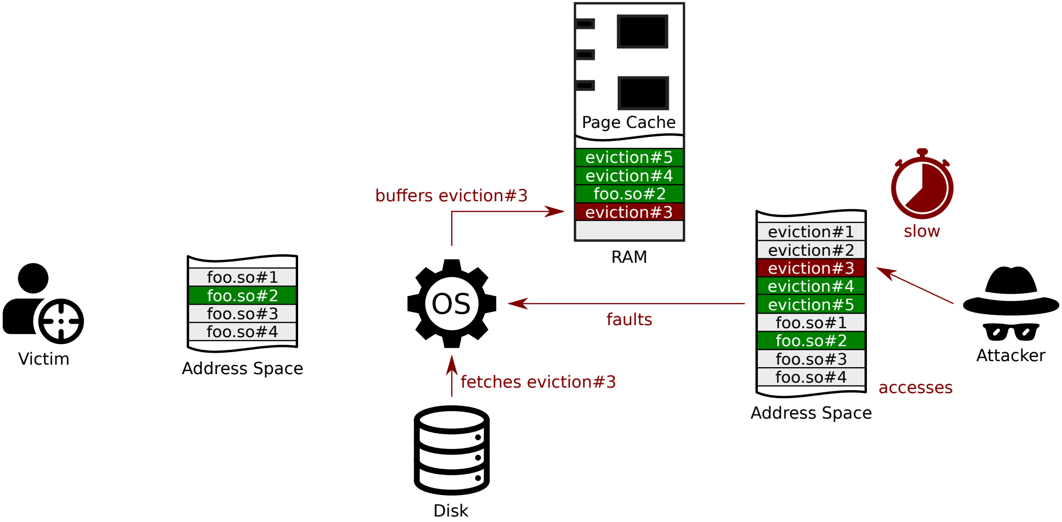
# Page Cache Attacks

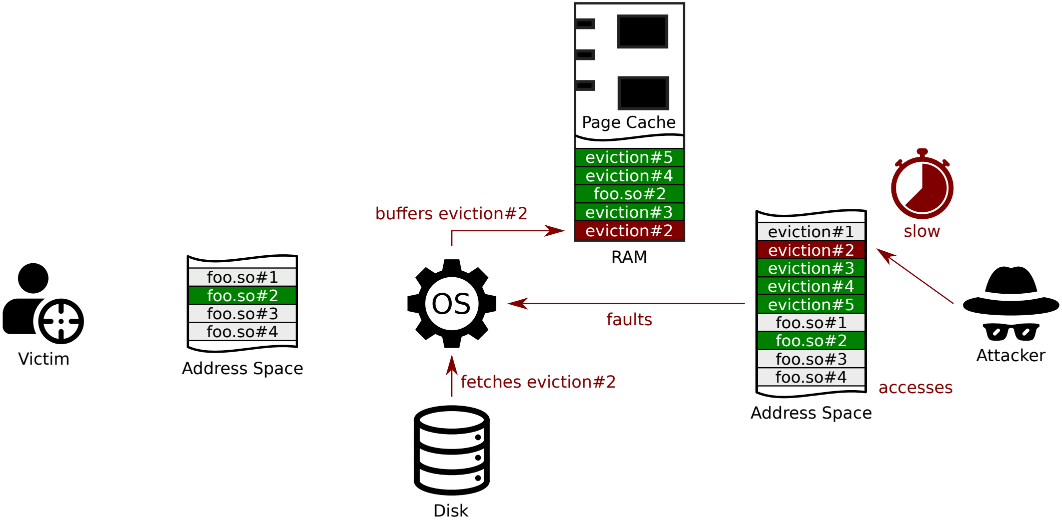


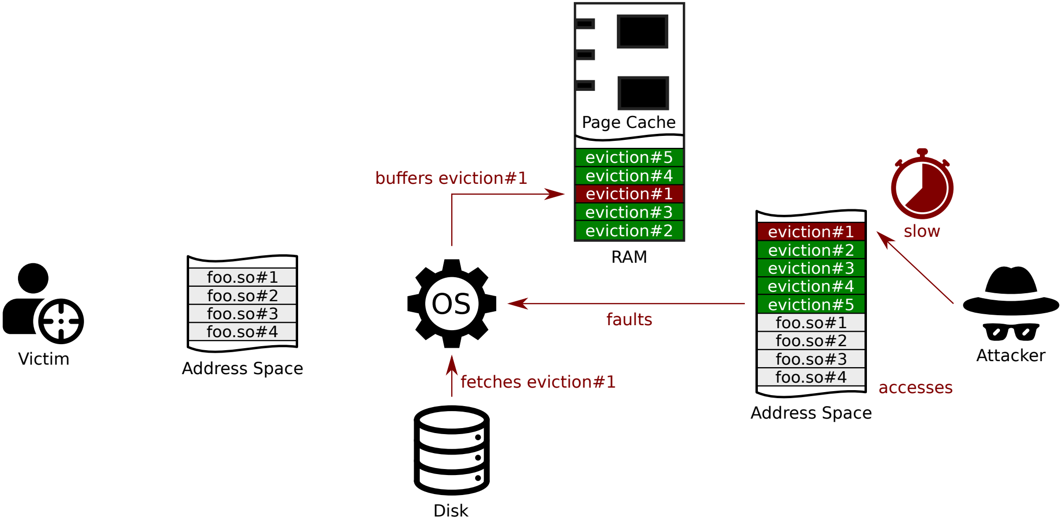
# Page Cache Attacks

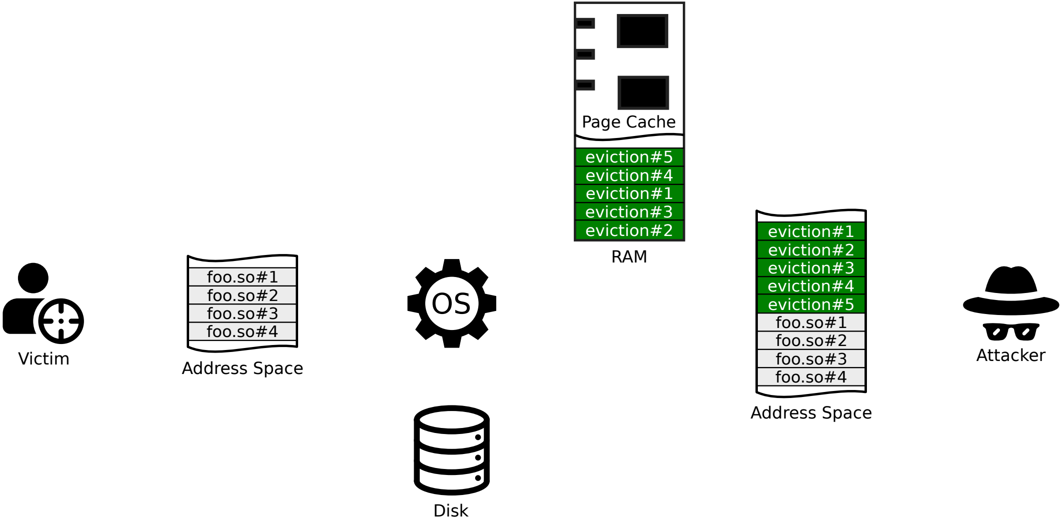


# Page Cache Attacks



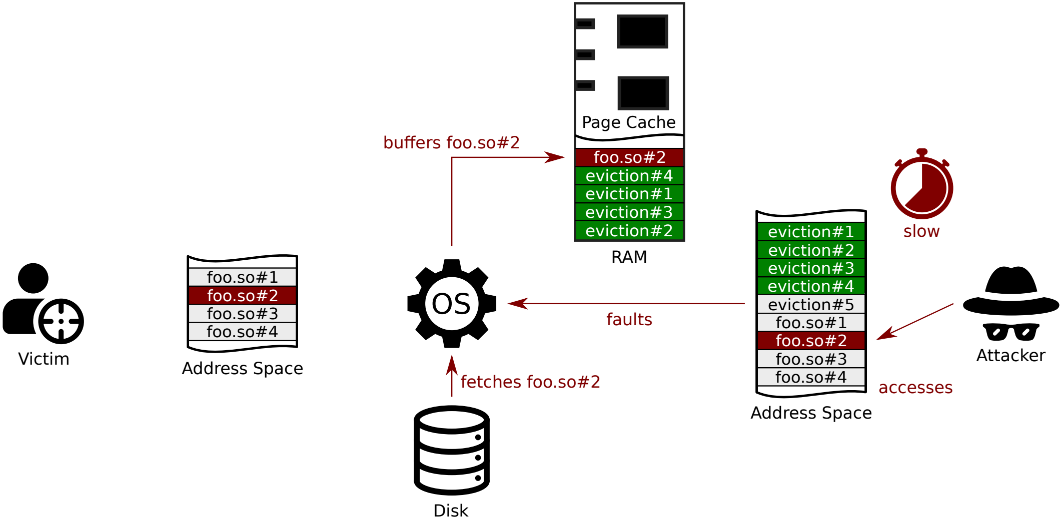

Page Cache
eviction#5
eviction#4
foo.so#2
eviction#3
eviction#2
RAM
buffers eviction#2
slow
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
OS
faults
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
Attacker
accesses
fetches eviction#2
Disk


Page Cache
eviction#5
eviction#4
eviction#1
eviction#3
eviction#2
RAM
buffers eviction#1
slow
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
accesses
Attacker
faults
OS
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
fetches eviction#1
Disk

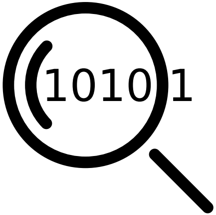

Page Cache
eviction#5
eviction#4
eviction#1
eviction#3
eviction#2
RAM
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
OS
Disk
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address Space
Attacker

# Page Cache Attacks

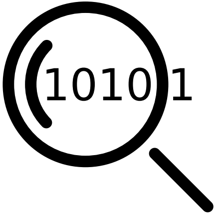





I AM
WATCHING YOU
imgflip.com



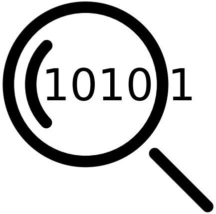

First idea:




First idea:

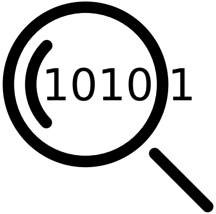
- Measure page access time

# Observe Page Cache State




First idea:

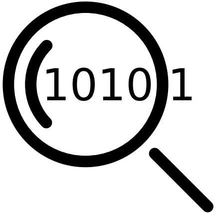
- Measure page access time
- Buffers pages in page cache → destructive




First idea:

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

# Observe Page Cache State

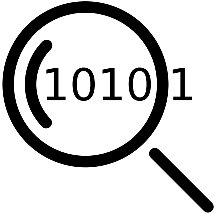



First idea:

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

Better:

# Observe Page Cache State

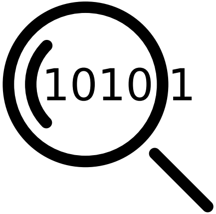



First idea:

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

Better:

- Use APIs provided by the operating system

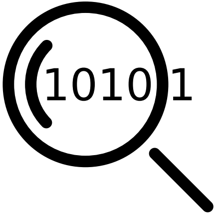



First idea:

- 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`



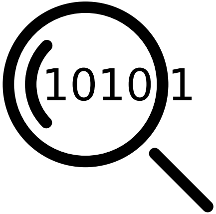

First idea:

- 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



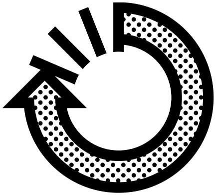

First idea:

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

Better:

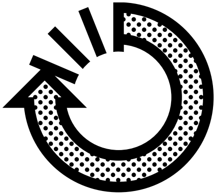
- Use APIs provided by the operating system
  - `mincore`
  - `QueryWorkingSetEx`
- Non-destructive → higher average resolution

## Reset Page Cache State



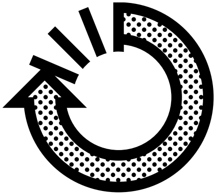
- Necessary for detecting multiple accesses

# Reset Page Cache State



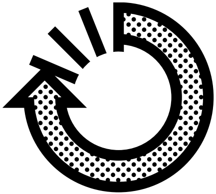
- Necessary for detecting multiple accesses
- Bottleneck of side channel

# Reset Page Cache State



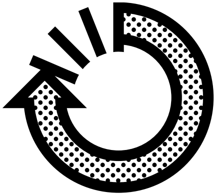
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on memory management implementation

# Reset Page Cache State

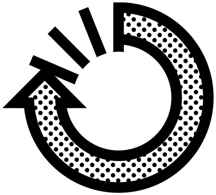


- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on memory management implementation
  - Differences in page replacement

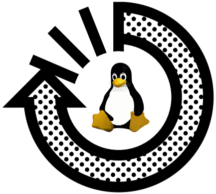
# Reset Page Cache State



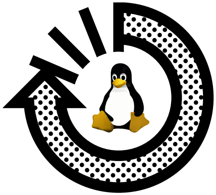
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on memory management implementation
  - Differences in page replacement
  - Global CLOCK-Pro like algorithm 



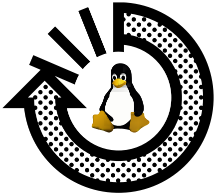
- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Ideal strategy depends on memory management implementation
  - Differences in page replacement
  - Global CLOCK-Pro like algorithm
  - Per-process working sets with Aging algorithm



- Access pages until target page is replaced

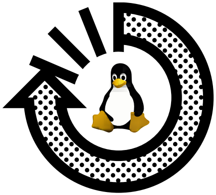


- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file



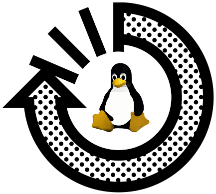
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- -O1: Add pages already in page cache

## Eviction Linux



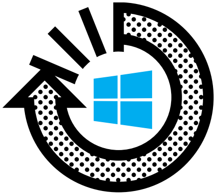
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- −O1: Add pages already in page cache
- −O2: Fill memory with anonymous dirty pages

# Eviction Linux



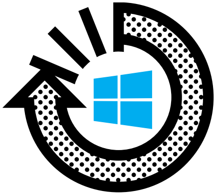
- Access pages until target page is replaced
- Basic eviction set: Large memory-mapped file
- -O1: Add pages already in page cache
- -O2: Fill memory with anonymous dirty pages
- Average run time down to 149 ms depending on optimisations

# Eviction Windows

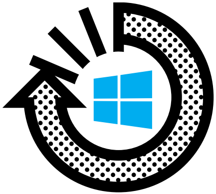


- Page cache eviction $\leftrightarrow$ target page drops out of all working sets

# Eviction Windows

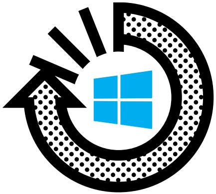


- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...



- 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)

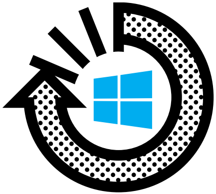
# Eviction Windows



- 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

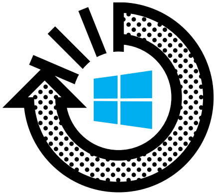
    $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)

# Eviction Windows



- Page cache eviction $\leftrightarrow$ 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)
    - for processes with same integrity level as attacker

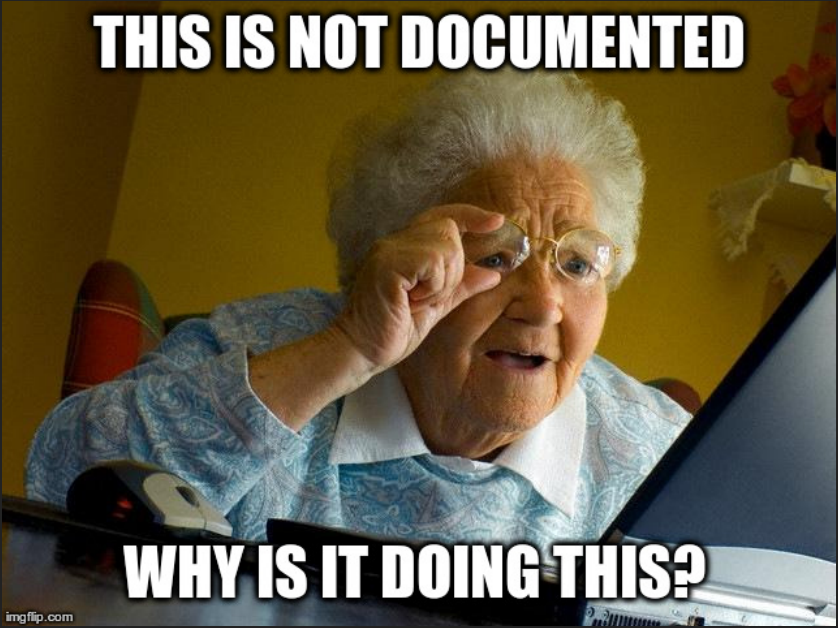
# Eviction Windows



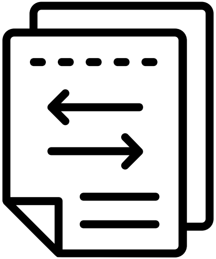
- Page cache eviction $\leftrightarrow$ target page drops out of all working sets
  - Previous approach slow...
- Optimizations:
  - Increase ws size + memory pressure $\rightarrow$ self-eviction ($<$2 s)
  - Evicting any page in other processes

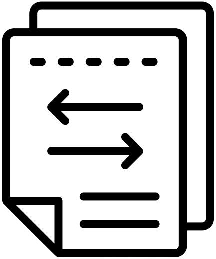
    $\rightarrow$ `SetProcessWorkingSetSize` (4.48 ms)
    - for processes with same integrity level as attacker
  - Evicting pages you have in your own working set

    $\rightarrow$ `VirtualUnlock` (17.69 $\mu$s)

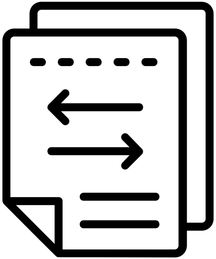

THIS IS NOT DOCUMENTED
WHY IS IT DOING THIS?
imgflip.com

- Shared file as information carrier

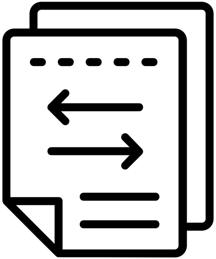




- Shared file as information carrier
- File page presence in page cache $\leftrightarrow$ message bits

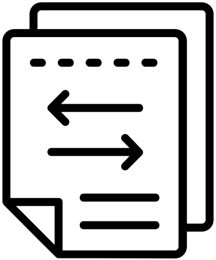


- Shared file as information carrier
- File page presence in page cache $\leftrightarrow$ message bits
- Additional file pages for transmission control



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

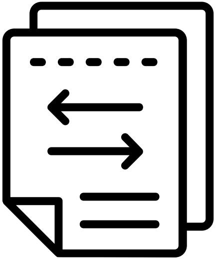
Different implementation approaches:



- 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 |
| Linux | `madvise` `posix_fadvise` | `mincore` | 81.16 kB/s |
| Windows | process WS `VirtualUnlock` | `QueryWorkingSetEx` (`ShareCount`) | 100.11 kB/s |



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

Different implementation approaches:

| OS | Eviction | Observation | Speed |
|---|---|---|---|
| Linux | like side channel | `mincore` | 20.20 kB/s |
| Linux | `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

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

- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks
    - Returns UNIX timestamp in microseconds

- 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

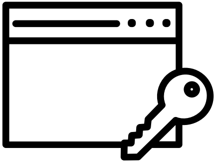
- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks
    - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to detect `microtime` call
    - Seed recoverable
- `zif_microtime` on page 781 of `php-fpm` executable

# PHP RNG Attack

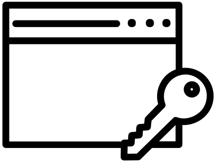
- Targets seeding of PHP PRNG
- `microtime` used as seed by some frameworks
  - Returns UNIX timestamp in microseconds
- Later PRNG used for cryptographic operations :(
- Side channel used to detect `microtime` call
  - Seed recoverable
- `zif_microtime` on page 781 of `php-fpm` executable
  - `PHP 7.3.5`,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

- 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: $\pm 1$ ms

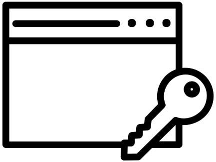
# Live Demo



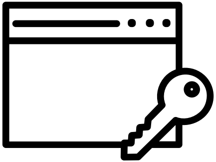
- Detect opening of interesting window



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

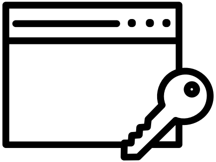


- Detect opening of interesting window
    - e.g. authentication windows
- Overlay original window with fake



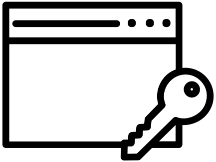
- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger

# UI Redressing Attack



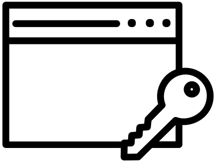
- Detect opening of interesting window
  - e.g. authentication windows
- Overlay original window with fake
- Side channel used as a trigger
- Provides low latency → hardly noticeable

# 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`

- Identified as CVE-2019-5489

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

# Impact

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


PATCHES
PATCHES EVERYWHERE
makeameme.org

- Higher privilege for `QueryWorkingSetEx` on other processes

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

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

# 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

- 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 - Linux

- `mincore` only reveals information for writeable file pages

- `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
  - Merged with the release of the 5.1.4 kernel

# 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?

# 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

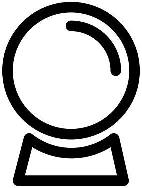
- We want the performance optimizations

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

# Countermeasures are Difficult

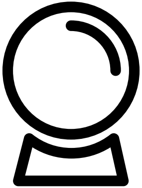
- We want the performance optimizations
- Many side-channel attacks exploit intended behavior
- Often a trade-off between security and performance

- 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



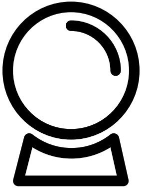
- We won't get rid of side channels

# 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 → 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

# 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.

# Page Cache Attacks

## Microarchitectural Attacks on Flawless Hardware

Daniel Gruss, Trishita Tiwari, Michael Schwarz, Erik Kraft