Software-based Microarchitectural Attacks

Daniel Gruss

July 8, 2018

Graz University of Technology

amazon.com
Prime+Probe

ANOTHER FLIP IN THE ROW

ROWHAMMER.JS

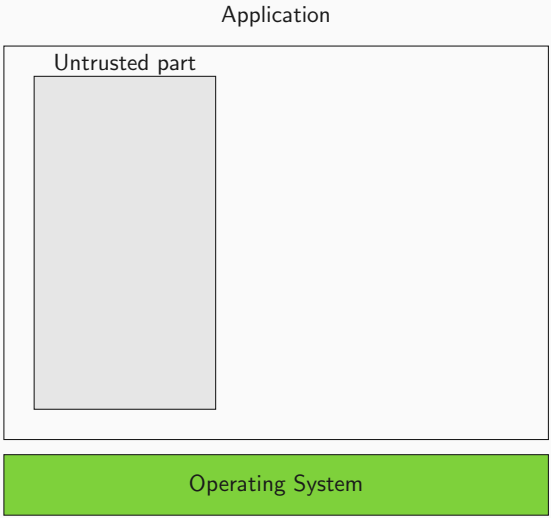
FANTASTIC TIMERS

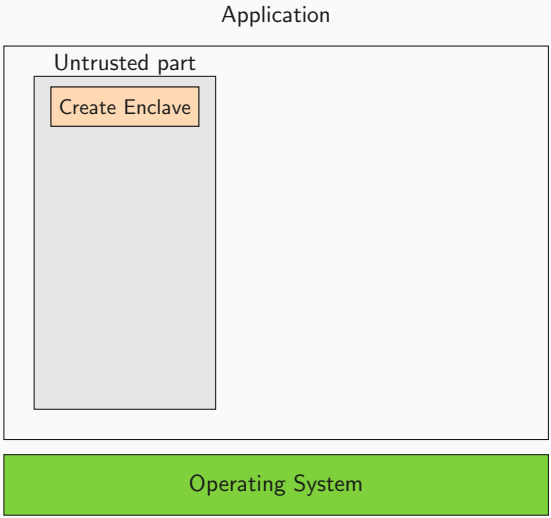
AND WHERE TO FIND THEM

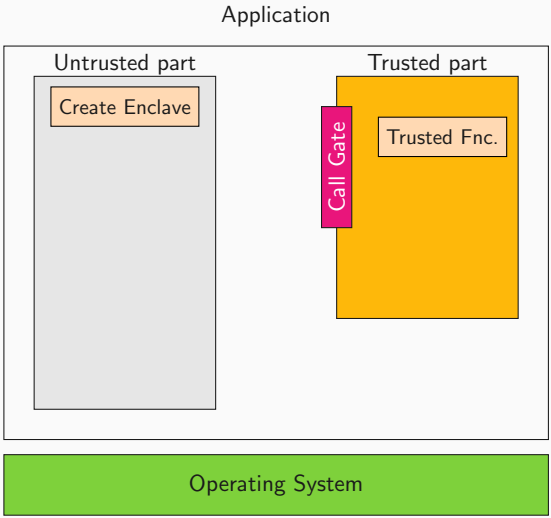
HIGH-RESOLUTION MICROARCHITECTURAL ATTACKS IN JAVASCRIPT

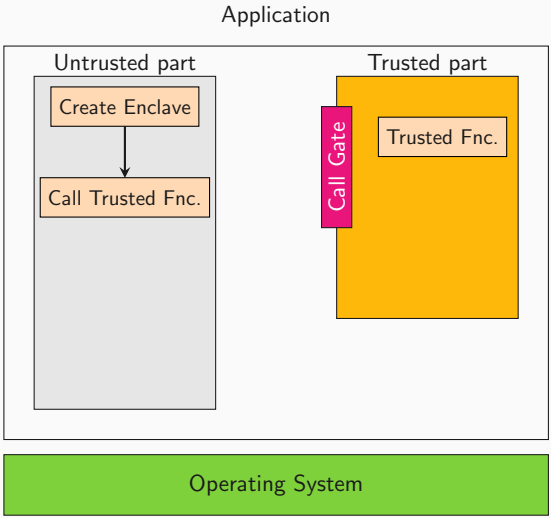
JavaScript zero

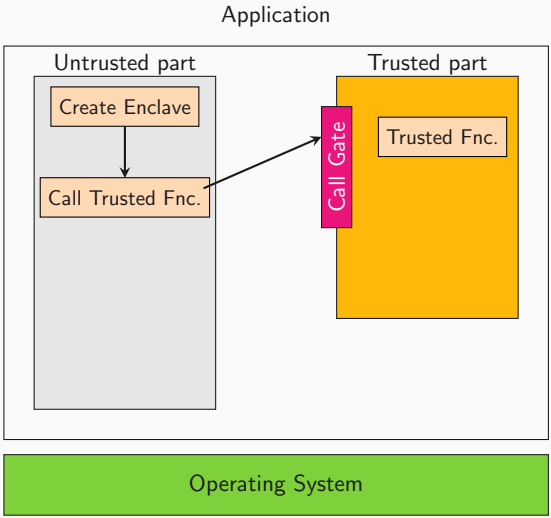
REAL JavaScript AND ZERO SIDE-CHANNEL ATTACKS

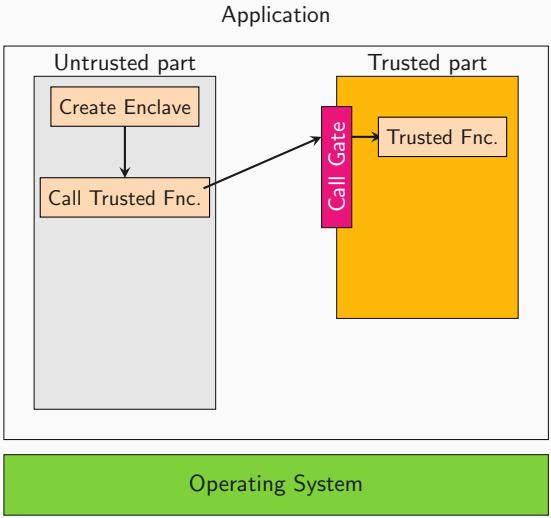

Application
Untrusted part
Operating System

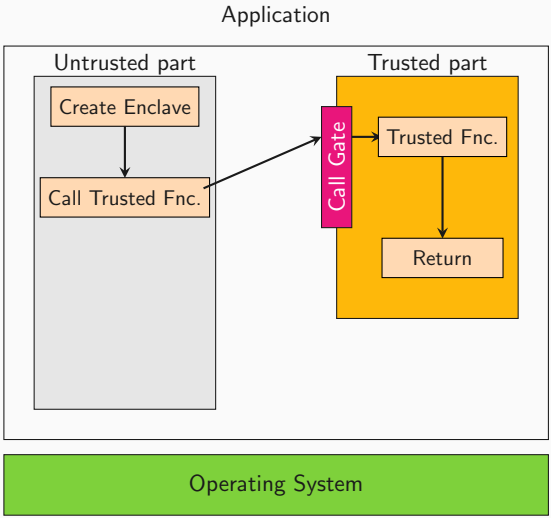

Application
Untrusted part
Create Enclave
Operating System

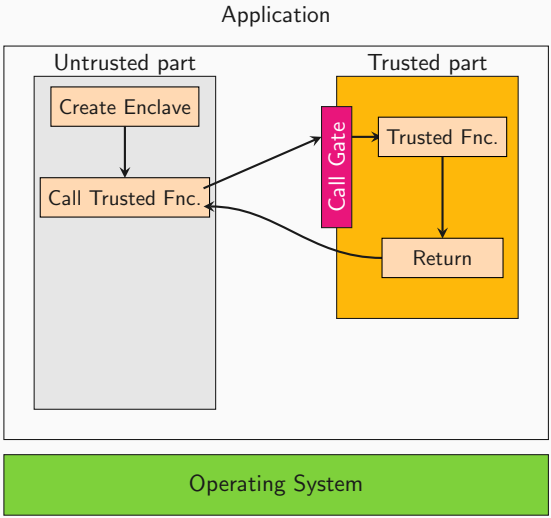

Application
Untrusted part
Create Enclave
Trusted part
Call Gate
Trusted Fnc.
Operating System

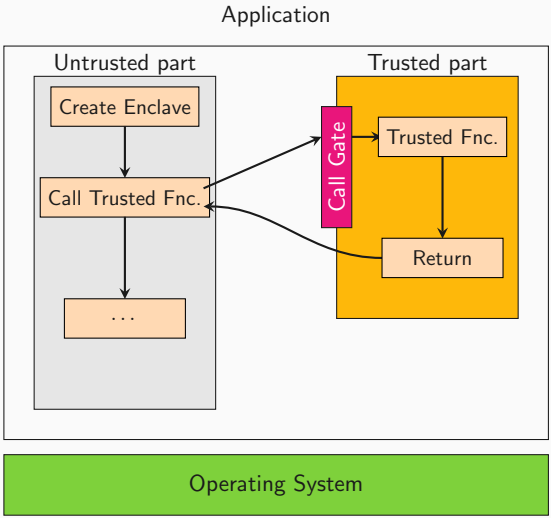

Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

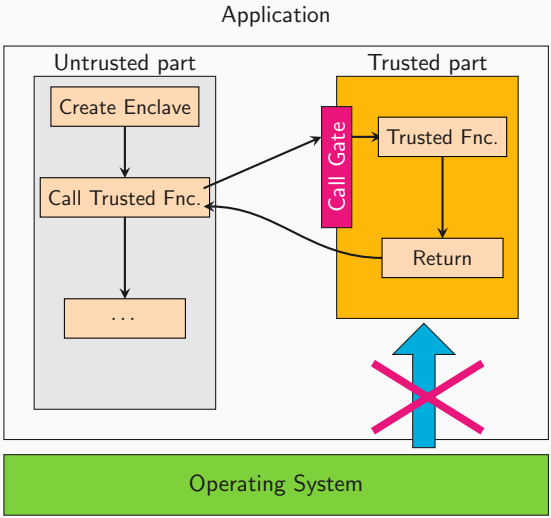

Application
Untrusted part
Create Enclave
Call Trusted Fnc.
Trusted part
Call Gate
Trusted Fnc.
Operating System

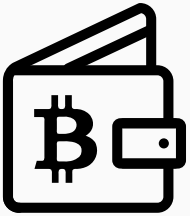

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System

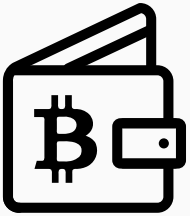

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
Call Gate
Trusted Fnc.
Return
Operating System


Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System

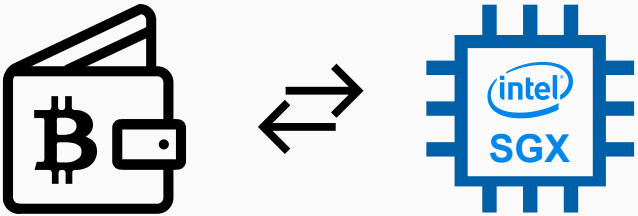

Application
Untrusted part
Trusted part
Create Enclave
Call Trusted Fnc.
...
Call Gate
Trusted Fnc.
Return
Operating System



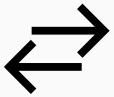
Stealing Bitcoins?

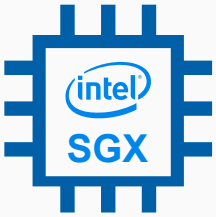


Stealing Bitcoins?

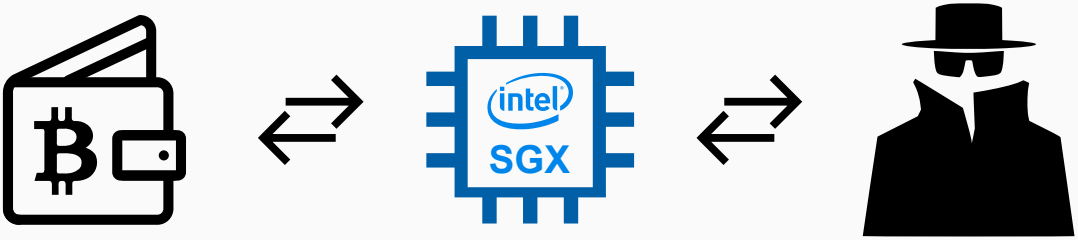


Stealing Bitcoins?





Stealing Bitcoins?



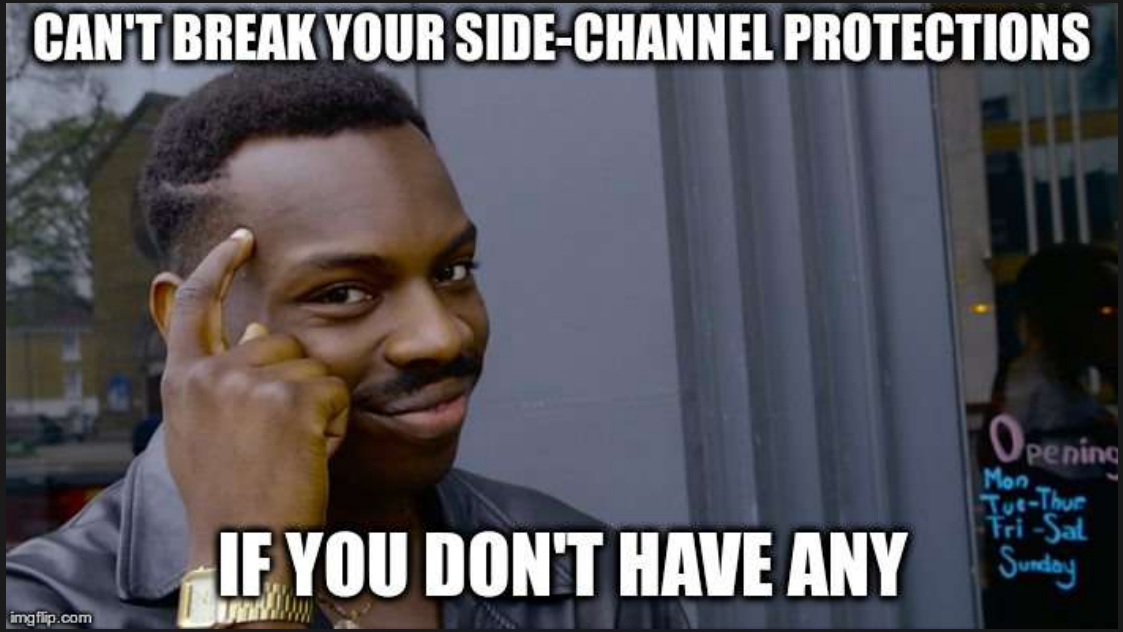
Protection from Side-Channel Attacks

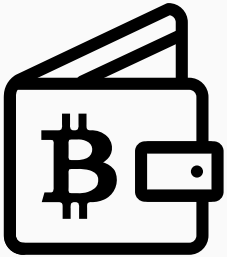
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks.

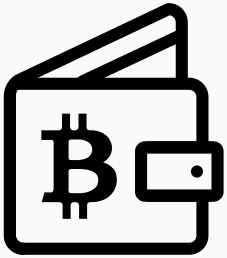
Protection from Side-Channel Attacks

Intel SGX does not provide explicit protection from side-channel attacks. It is the enclave developer's responsibility to address side-channel attack concerns.


CAN'T BREAK YOUR SIDE-CHANNEL PROTECTIONS
IF YOU DON'T HAVE ANY
Opening
Mon
Tue-Thur
Fri-Sat
Sunday
imgflip.com



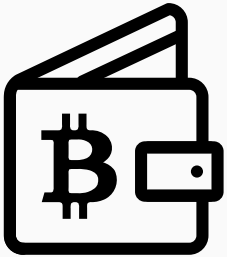
- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX



- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

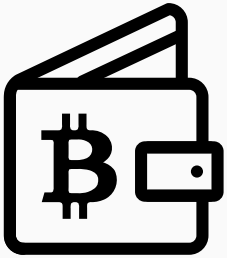
[...] We assume the TEE guarantees to hold



- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE.

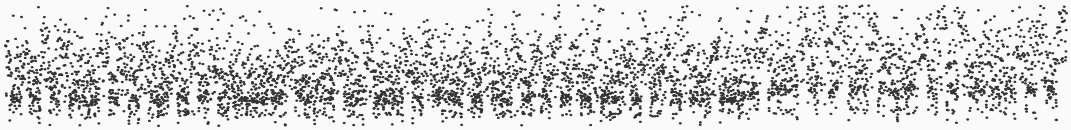


- Ledger SGX Enclave for blockchain applications
- BitPay Copay Bitcoin wallet
- Teechain payment channel using SGX

Teechain

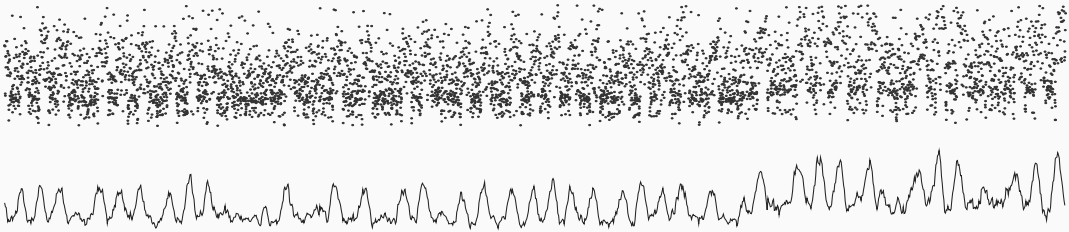
[...] We assume the TEE guarantees to hold and do not consider side-channel attacks [5, 35, 46] on the TEE. Such attacks and their mitigations [36, 43] are outside the scope of this work. [...]

Raw Prime+Probe trace...[1]



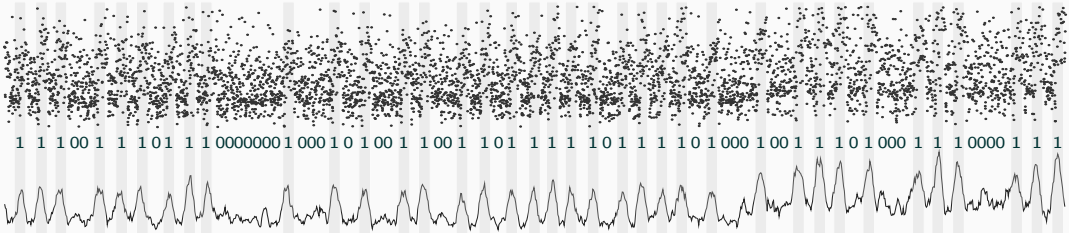
[1] M. Schwarz, D. Gruss, S. Weiser, C. Maurice, and S. Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...processed with a simple moving average...[2]

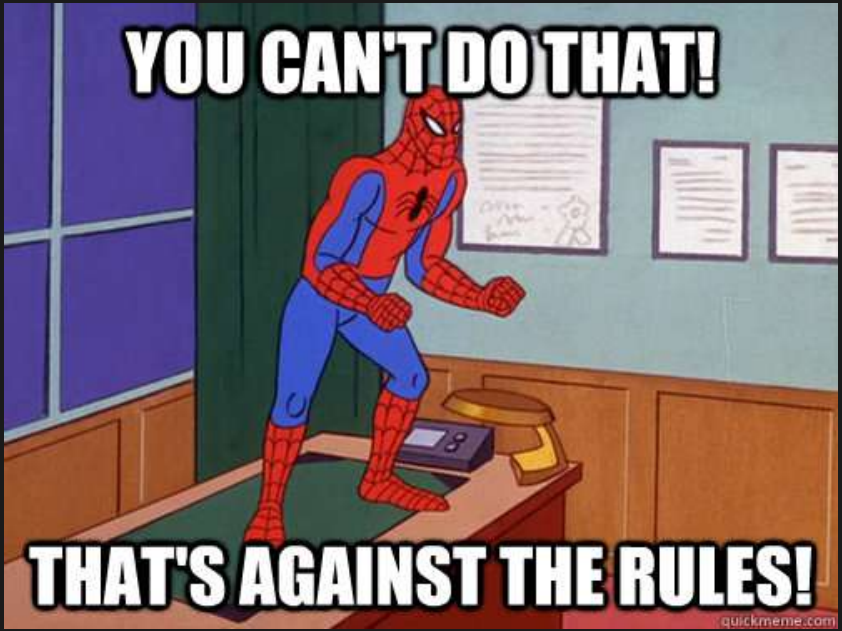


[2] M. Schwarz, D. Gruss, S. Weiser, C. Maurice, and S. Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.

...allows to clearly see the bits of the exponent[3]



[3] M. Schwarz, D. Gruss, S. Weiser, C. Maurice, and S. Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.


YOU CAN'T DO THAT!
THAT'S AGAINST THE RULES!
quickmeme.com


WANT TO DISCUSS THREAT MODELS NOW?
38
imgflip.com

- Usually no physical access

- Usually no physical access
- Local code

- Usually no physical access
- Local code
- Co-located code

- Usually no physical access
- Local code
- Co-located code
- Different meanings of “remote”

- Usually no physical access
- Local code
- Co-located code
- Different meanings of “remote”
  1. Attacker controls code in browser sandbox (e.g., [Ore+15; GMM16])

- Usually no physical access
- Local code
- Co-located code
- Different meanings of “remote”
  1. Attacker controls code in browser sandbox (e.g., [Ore+15; GMM16])
  2. Attacker cannot control any code on the system

TIMING IS EVERYTHING

1337 4242

FOOD CACHE

Revolutionary concept!

Store your food at home, never go to the grocery store during cooking.

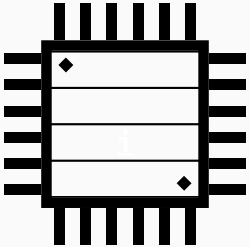
Can store **ALL** kinds of food.

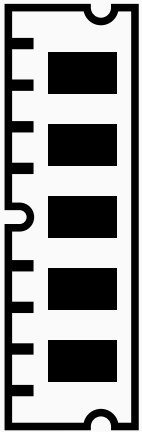
ONLY TODAY INSTEAD OF ~~$1,300~~

$1,299

ORDER VIA PHONE: +555 12345

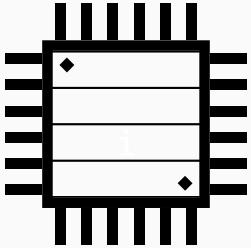
```
printf("%d", i);
printf("%d", i);
```

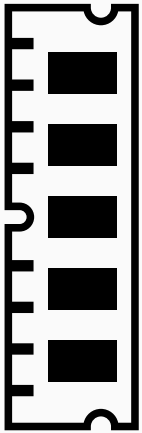


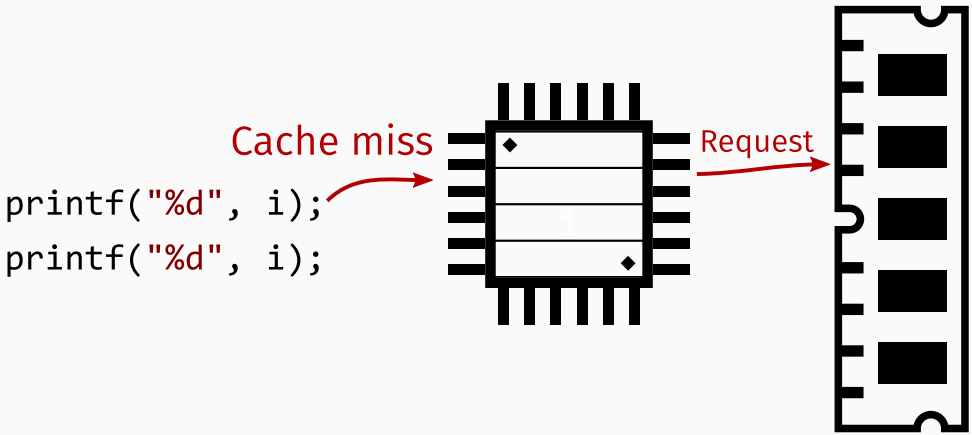


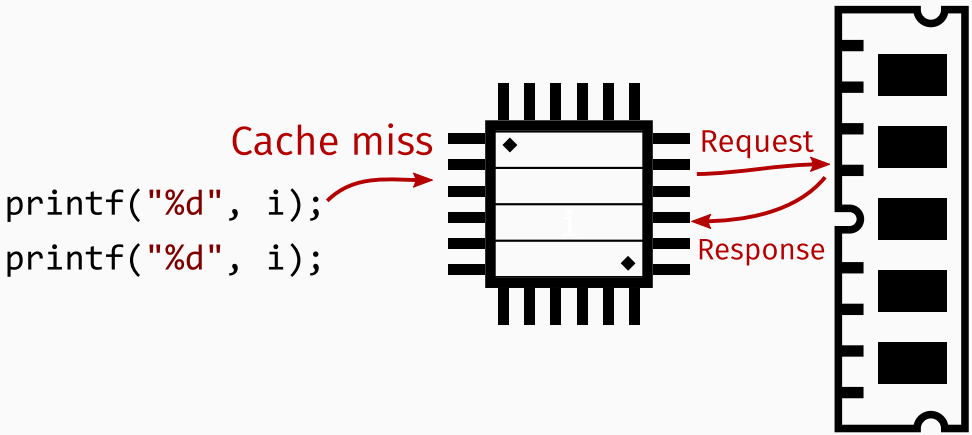
Cache miss

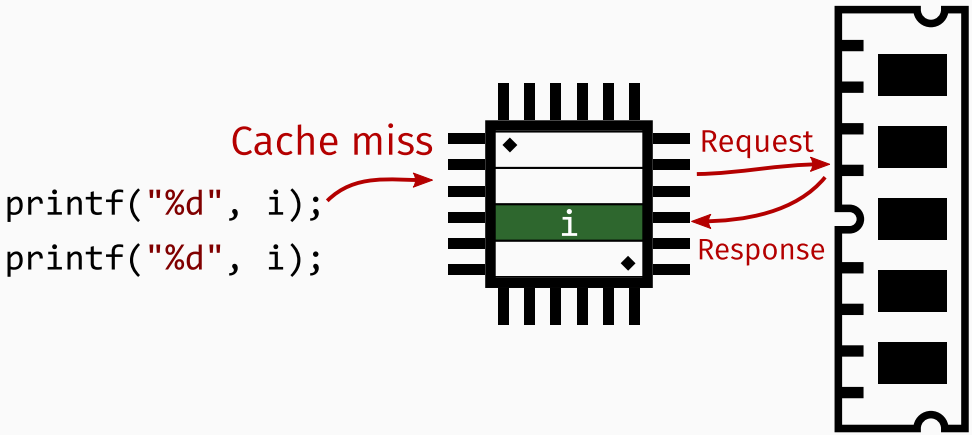
```
printf("%d", i);
printf("%d", i);
```

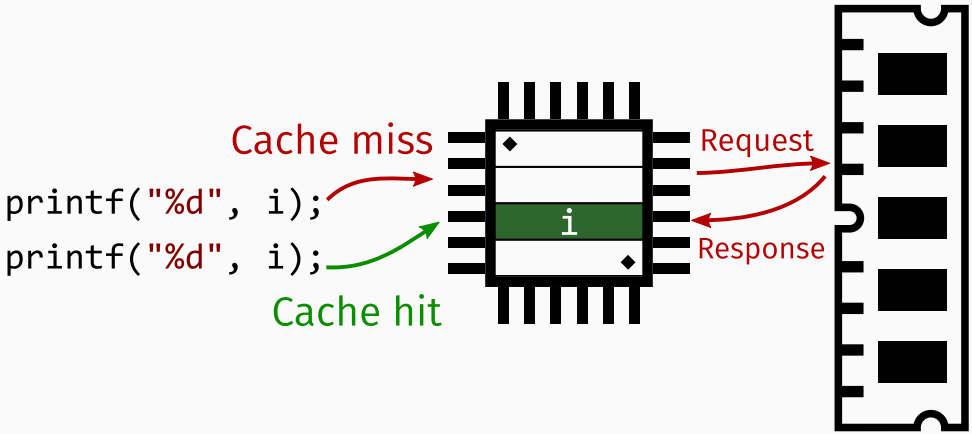


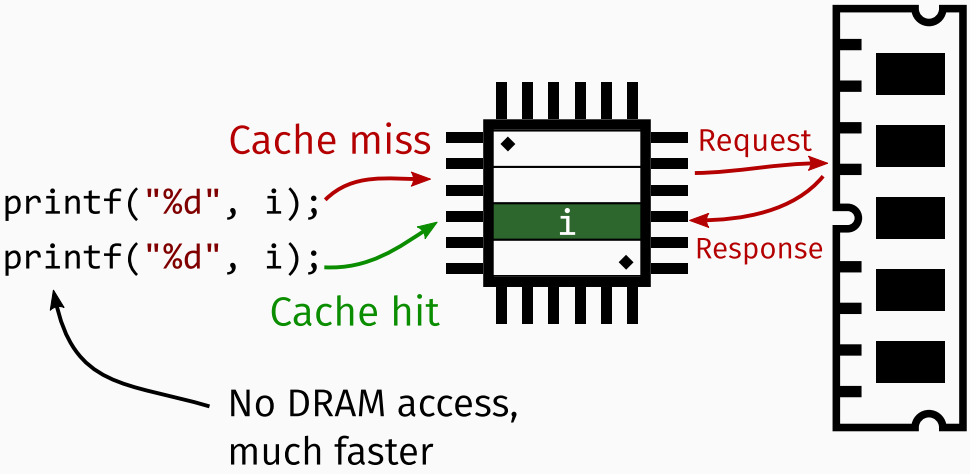


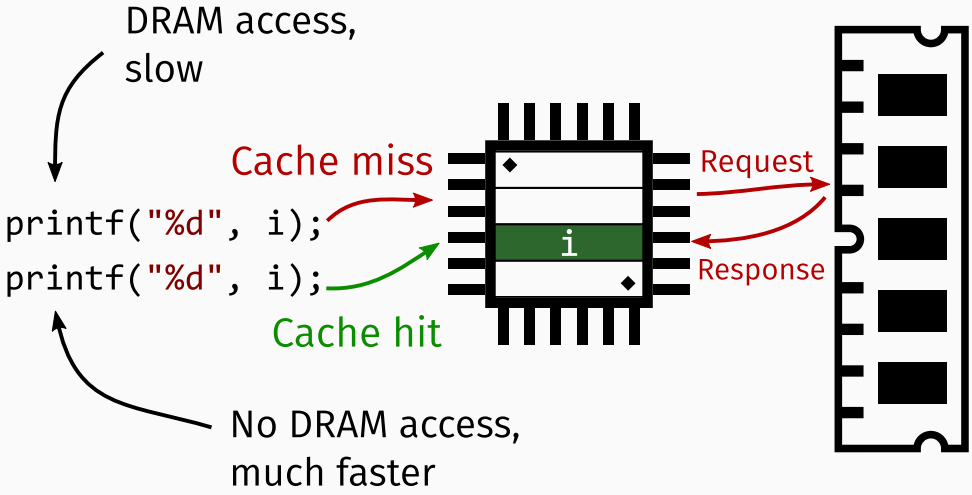

Cache miss
Request
printf("%d", i);
printf("%d", i);


Cache miss
printf("%d", i);
printf("%d", i);
Request
Response


Cache miss
printf("%d", i);
printf("%d", i);
i
Request
Response


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


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

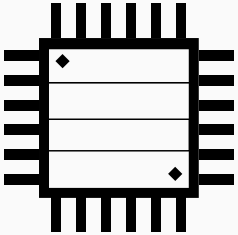

DRAM access,
slow
Cache miss
printf("%d", i);
printf("%d", i);
Cache hit
No DRAM access,
much faster
i
Request
Response

Shared Memory

ATTACKER

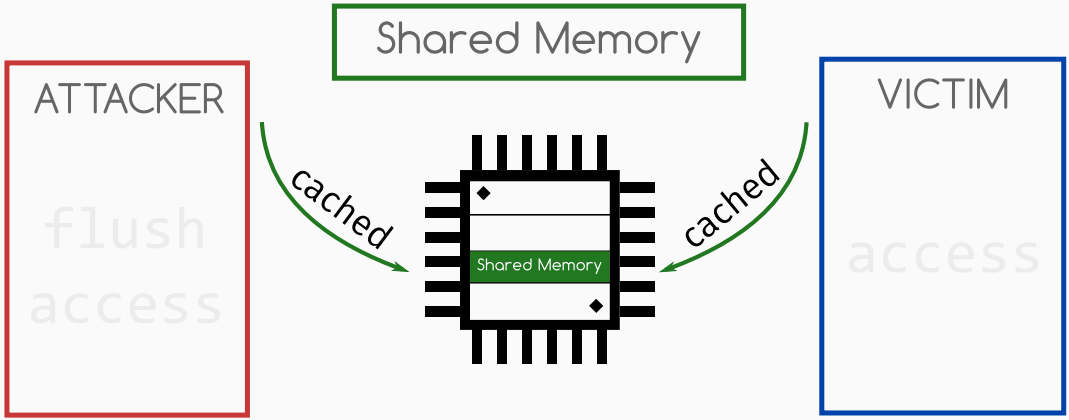
flush

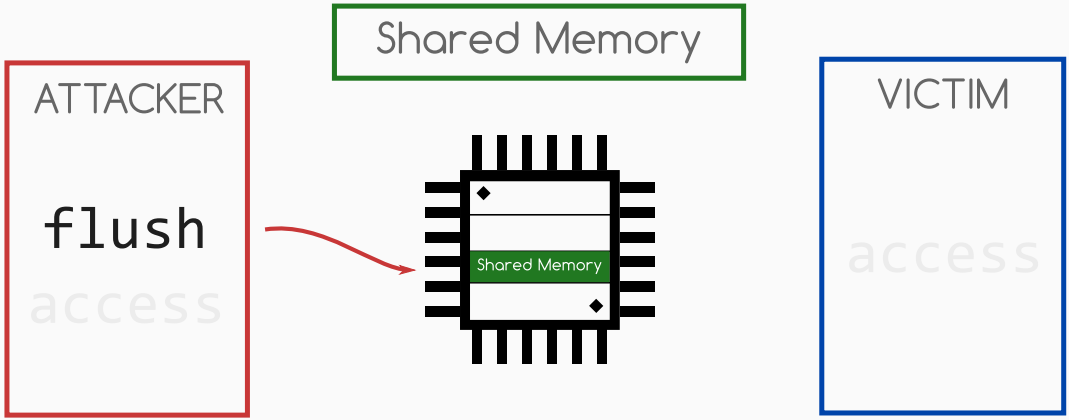
access



VICTIM

access


Shared Memory
ATTACKER
flush
access
cached
Shared Memory
cached
VICTIM
access

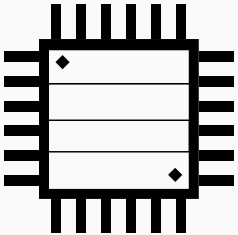

Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access

Shared Memory

ATTACKER

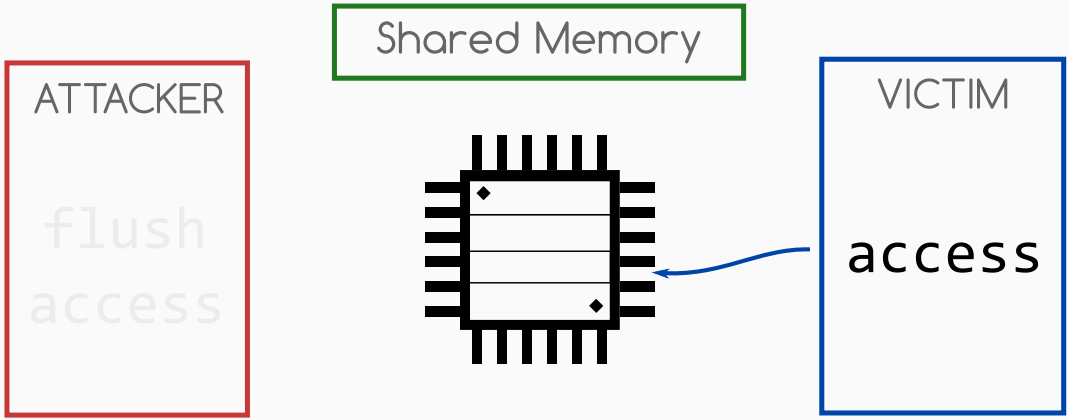
flush

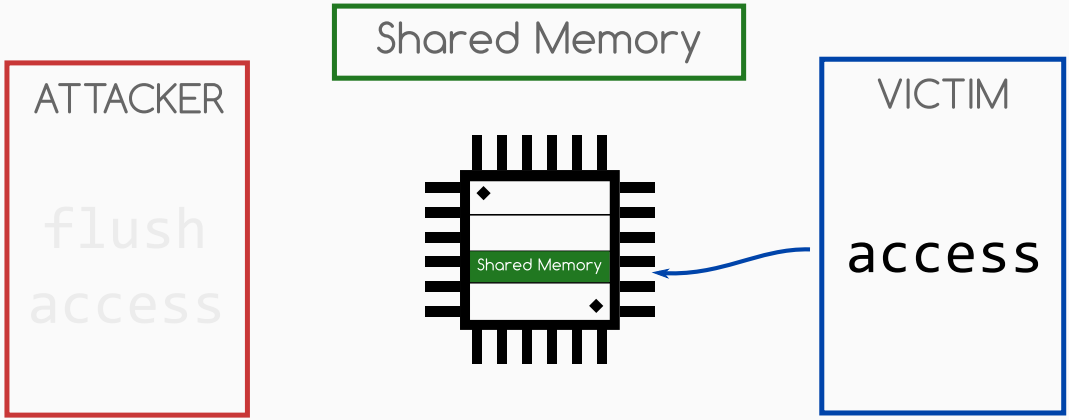
access

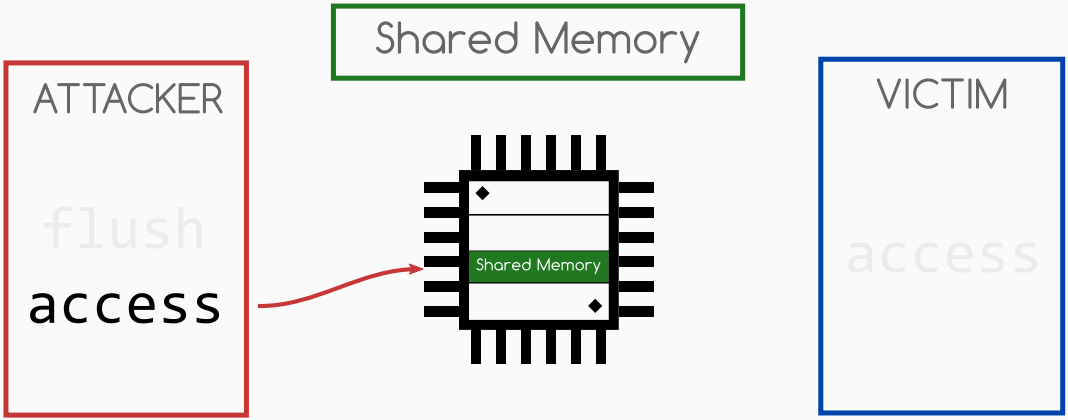


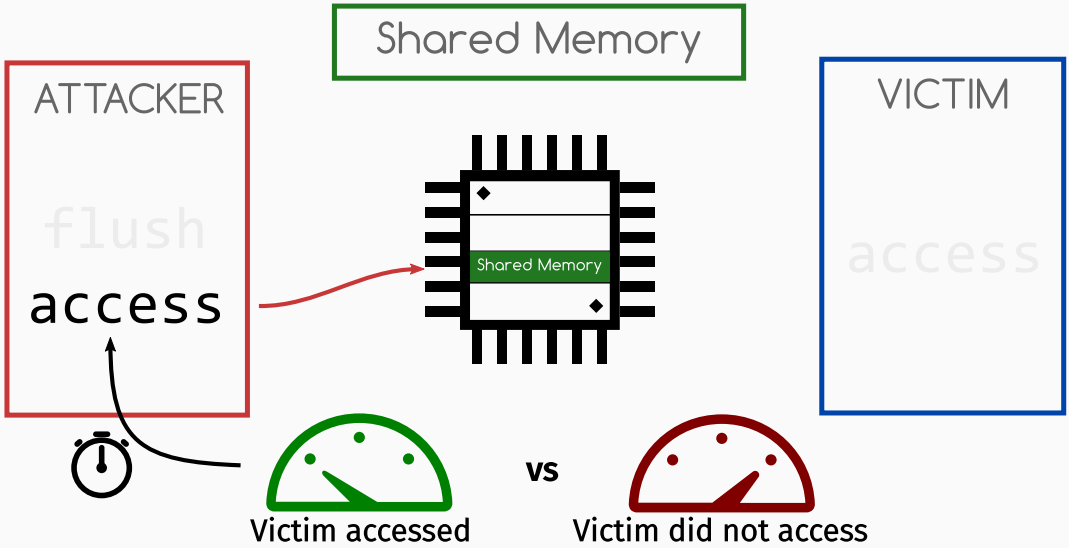
VICTIM

access


Shared Memory
ATTACKER
flush
access
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access


Shared Memory
ATTACKER
flush
access
Shared Memory
VICTIM
access
vs
Victim accessed
Victim did not access

Physical Side Channels

Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s

Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks

Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks

- often around nanoseconds

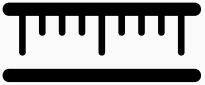
Physical Side Channels

- theoretical maximum accuracy of $5.4 \cdot 10^{-44}$s
- feasible today: $850 \cdot 10^{-21}$s

Microarchitectural Attacks

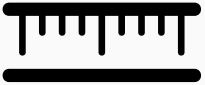
- often around nanoseconds
- sometimes much lower

Physical Side Channels

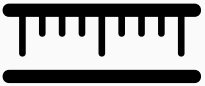


Physical Side Channels

- in the range of multiple GHz



Temporal Component: Sampling Rate

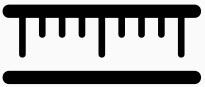


Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

Temporal Component: Sampling Rate

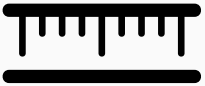


Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)

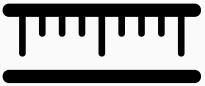


Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)

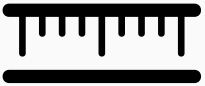


Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack

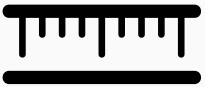


Physical Side Channels

- in the range of multiple GHz

Microarchitectural Attacks

- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack
  - device under test = measurement device



Physical Side Channels

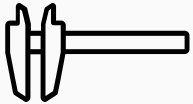
- in the range of multiple GHz

Microarchitectural Attacks

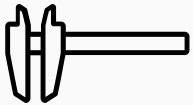
- usually varying frequency (depending on the attack)
- between a few ns (< 1 GHz) and multiple seconds (< 1 Hz) (or even worse)
- strongly dependent on the specific attack
  - device under test = measurement device
  - observer effect

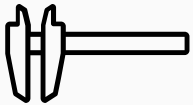
device under test = measurement device

- measuring time takes some time
- limits the resolution
- measuring cache hits/misses manipulates the cache state
- virtually all measurements are destructive



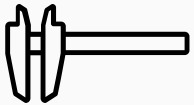
Flush+Reload has no noise except for:





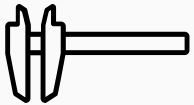
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)



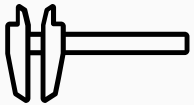
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution



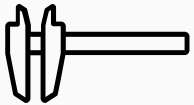
Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching



Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching
- ...



Flush+Reload has no noise except for:

- Race condition between attacker and victim (observer effect)
- Speculative execution
- Prefetching
- ...

$\rightarrow$ Typically $> 99.99\%$ precision and recall

Measuring Processor Operations

- Very short timings
- `rdtsc` instruction: “cycle-accurate” timestamps

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

What are we measuring?

- Do you measure what you *think* you measure?
- *Out-of-order* execution → what is really executed

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

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

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


FAIL
failblog.org

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

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

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

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

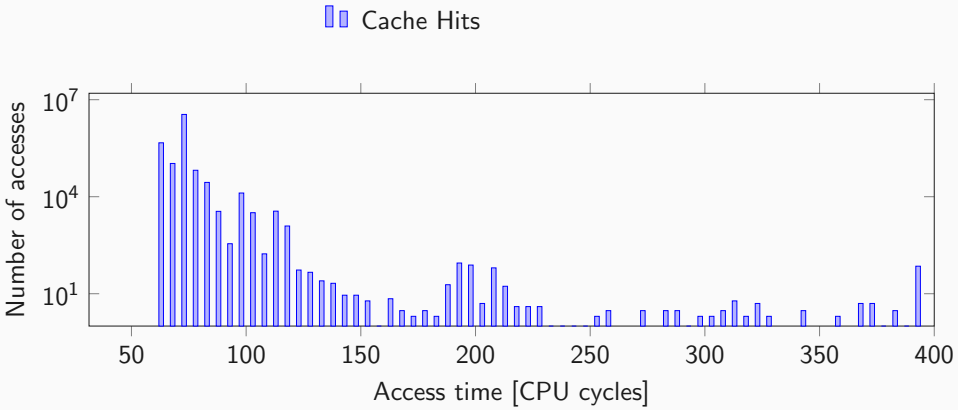
Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

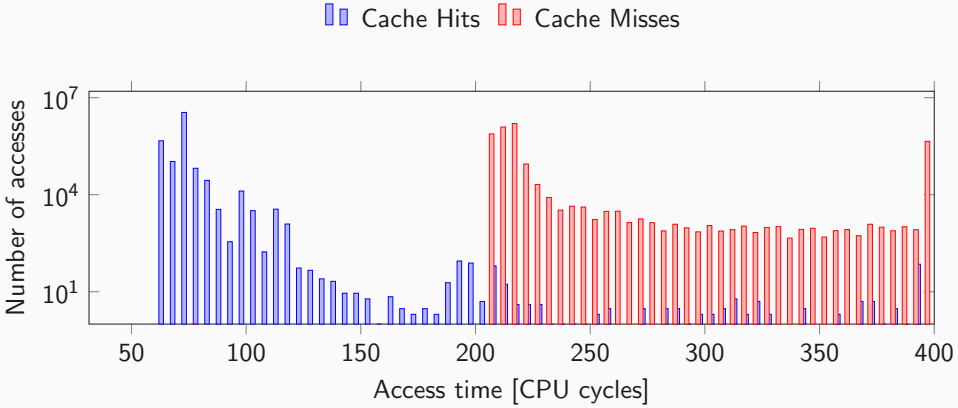
AUGUST 22, 2018 BY BRUCE

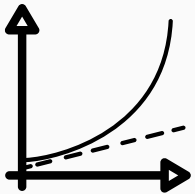
Intel Publishes Microcode Security Patches, No Benchmarking Or Comparison Allowed!

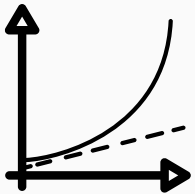
UPDATE: **Intel has resolved their microcode licensing issue which I complained about in this blog post.** The new license text is here.

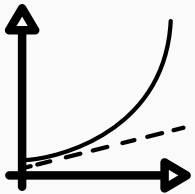
Memory Access Latency



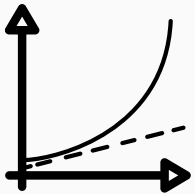

Cache Hits
Cache Misses
Number of accesses
10^7
10^4
10^1
50
100
150
200
250
300
350
400
Access time [CPU cycles]



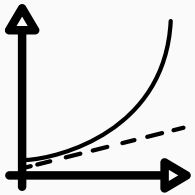




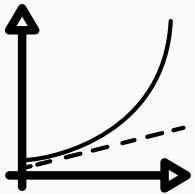
- Flush+Reload had beautifully nice timings, right?



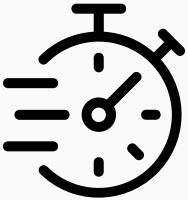
- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles



- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
  - only 35-70 steps between hits and misses

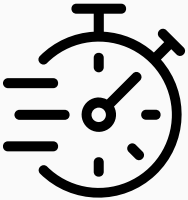


- Flush+Reload had beautifully nice timings, right?
- Well... steps of 2-4 cycles
  - only 35-70 steps between hits and misses
- On some devices only 1-2 steps!



- We can build our own timer [Lip+16; Sch+17]

- We can build our own timer [Lip+16; Sch+17]
- Start a thread that continuously increments a global variable



- We can build our own timer [Lip+16; Sch+17]
- Start a thread that continuously increments a global variable
- The global variable is our timestamp


ARE YOU REALLY EXPECTING TO
OUTPERFORM THE HARDWARE COUNTER?
imgflip.com

CPU cycles one increment takes

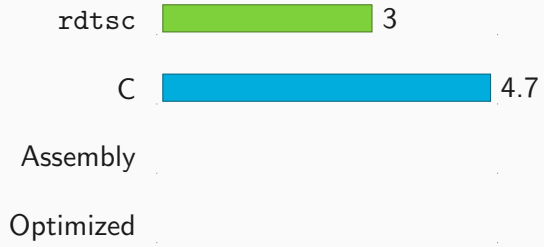

```
1 timestamp = rdtsc();
```

CPU cycles one increment takes


```
1 while(1) {
2   timestamp++;
3 }
```

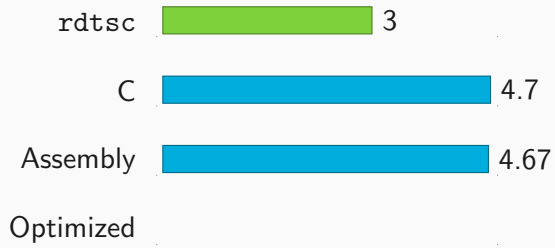
CPU cycles one increment takes

CPU cycles one increment takes

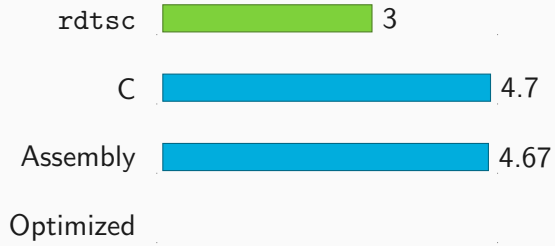



```
1 mov &timestamp, %rcx
2 1: incl (%rcx)
3 jmp 1b
```

CPU cycles one increment takes

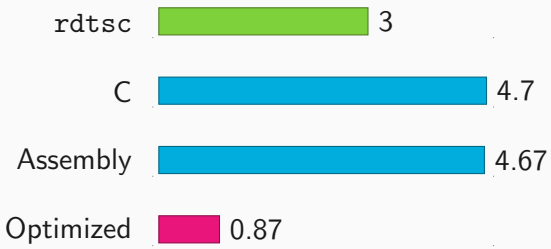


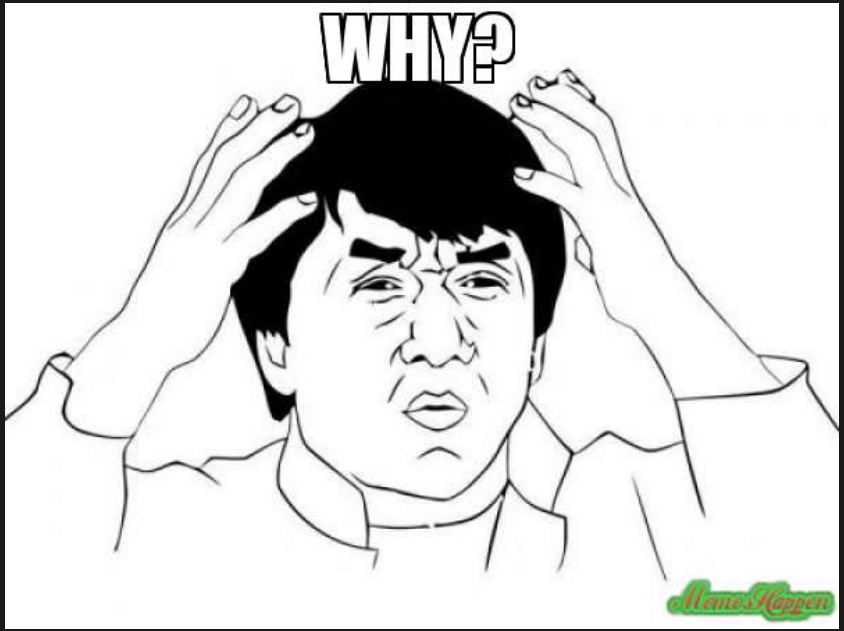
CPU cycles one increment takes

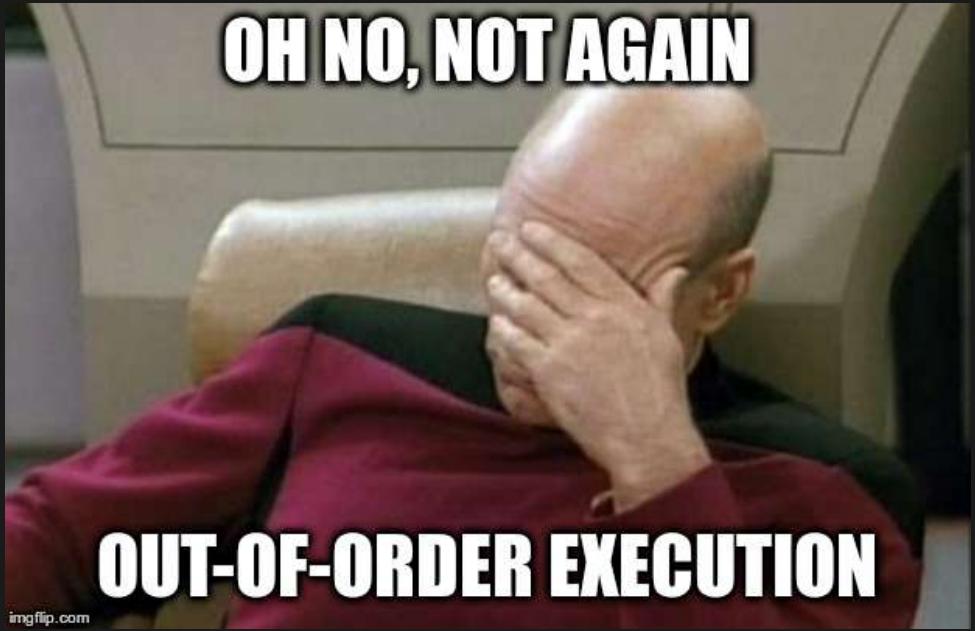


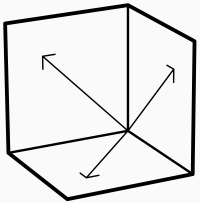

```
1 mov &timestamp, %rcx
2 1: inc %rax
3 mov %rax, (%rcx)
4 jmp 1b
```

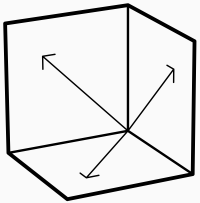
CPU cycles one increment takes

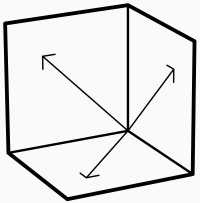



WHY?
MemeHappen

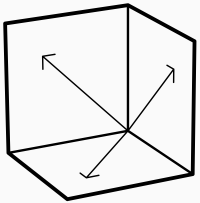

OH NO, NOT AGAIN
OUT-OF-ORDER EXECUTION
imgflip.com



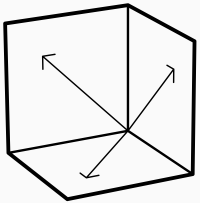




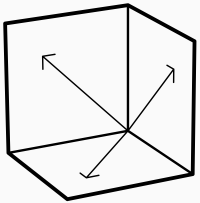
- physical: different offsets on the chip



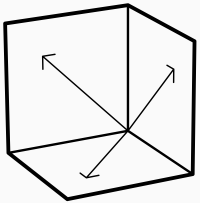
- physical: different offsets on the chip
- microarchitectural:



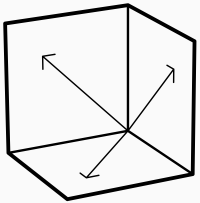
- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements



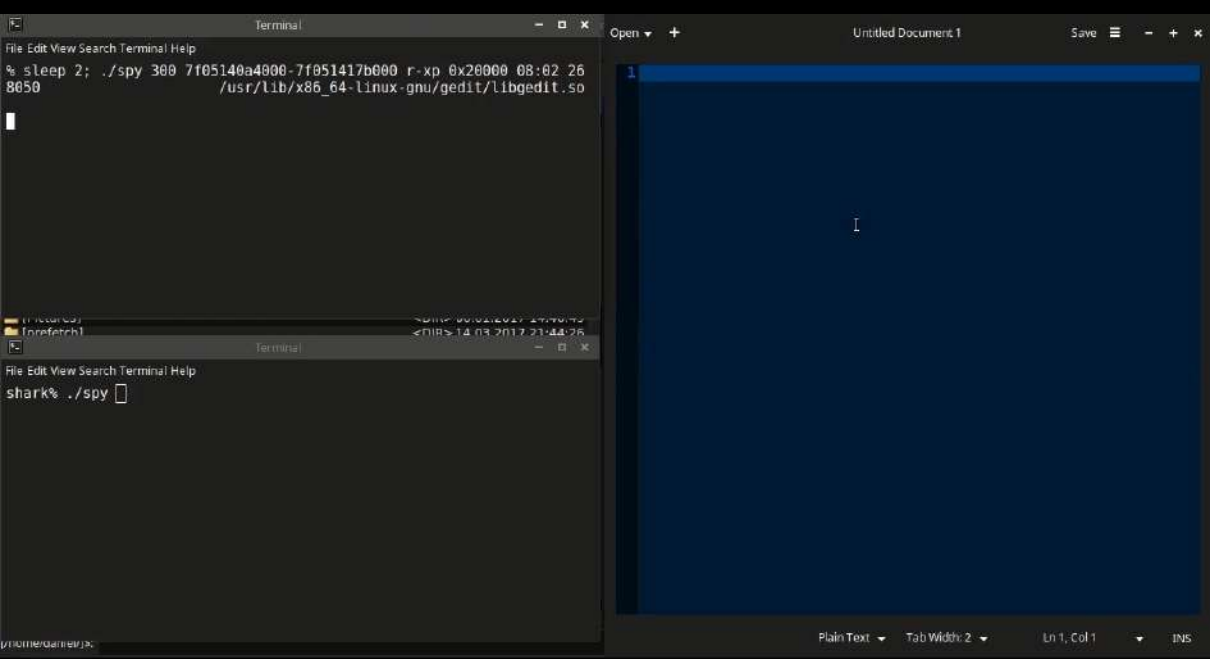
- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual adress space



- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual adress space
  - 2^{48} different virtual memory locations

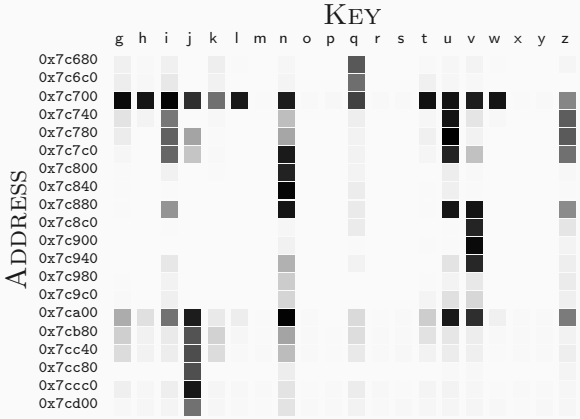


- physical: different offsets on the chip
- microarchitectural:
  - different microarchitectural elements
  - more significant: huge virtual adress space
  - 2^{48} different virtual memory locations
  - the location is often (part of) the secret



Cache Template Attack Demo

Cache Template[4]



[4] D. Gruss, R. Spreitzer, and S. Mangard. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In: USENIX Security Symposium. 2015.

- Managed by operating system

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

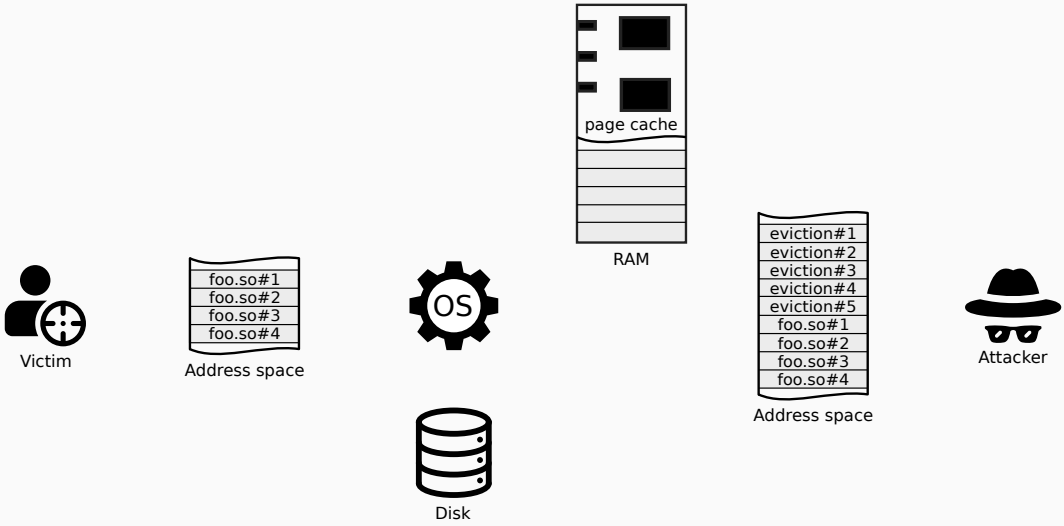
- Managed by operating system
- Buffers pages in RAM for faster accesses
- State of pages is tracked:

- Managed by operating system
- Buffers pages in RAM for faster accesses
- State of pages is tracked:
  - No write access $\rightarrow$ clean $\rightarrow$ no write back

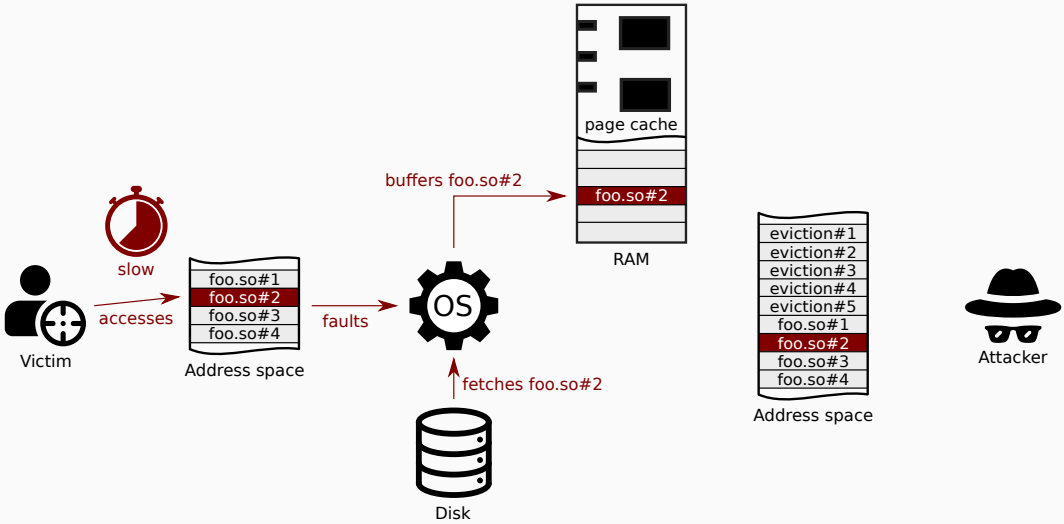
- Managed by operating system
- Buffers pages in RAM for faster accesses
- State of pages is tracked:
  - No write access → clean → no write back
  - Write access → dirty → write back

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

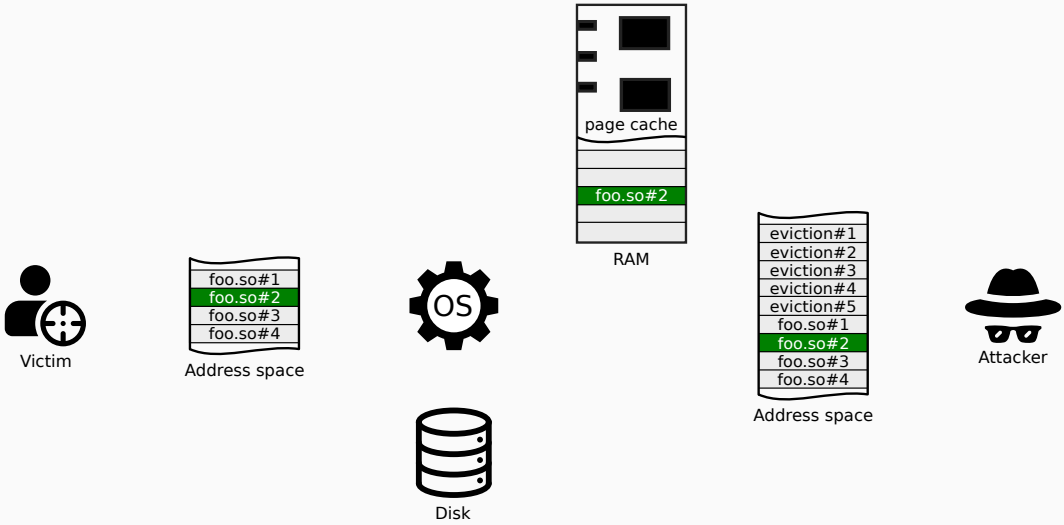
Page Cache Attacks



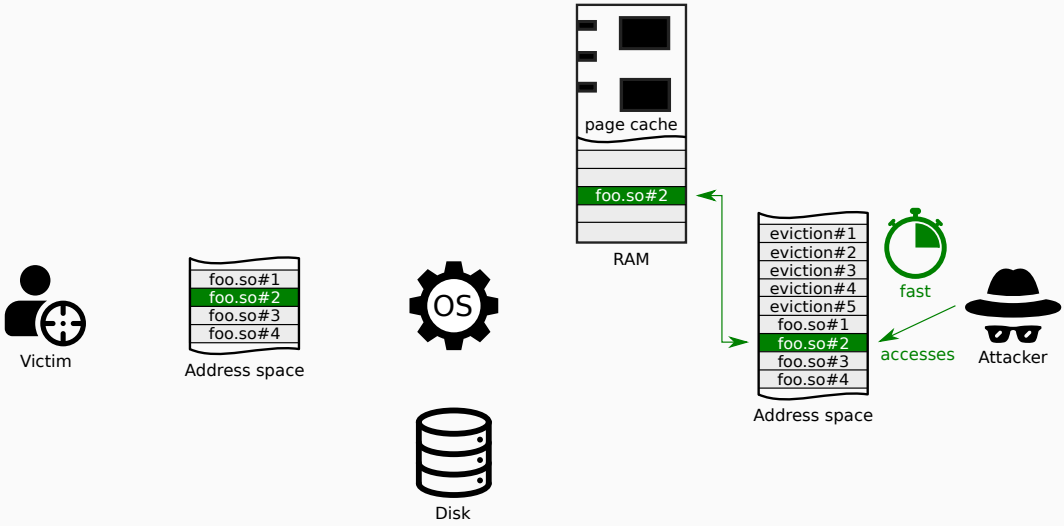
Page Cache Attacks



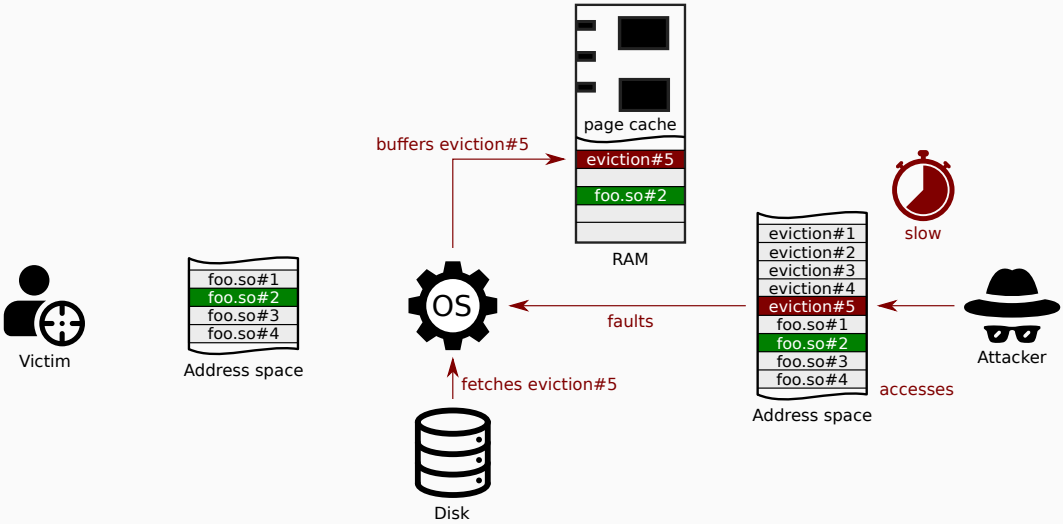
Page Cache Attacks

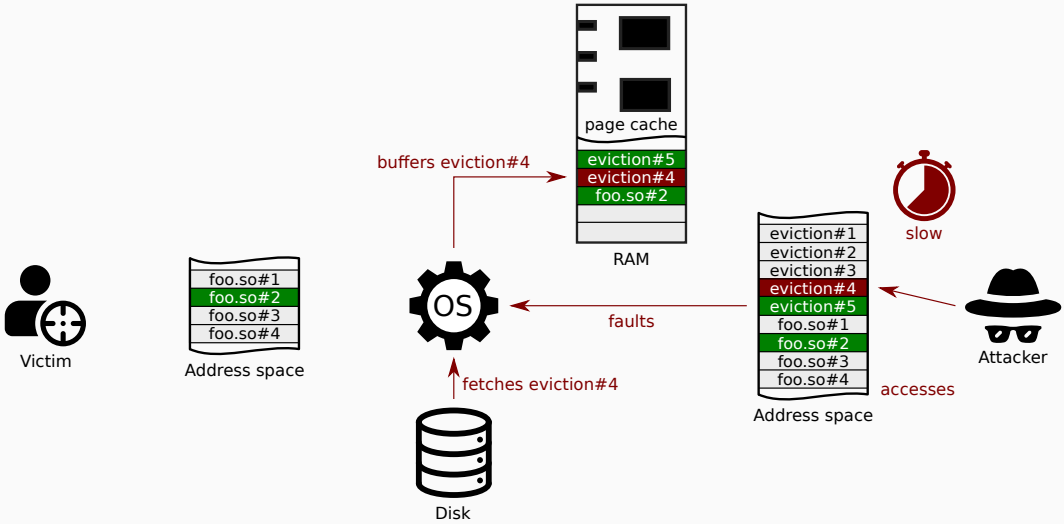


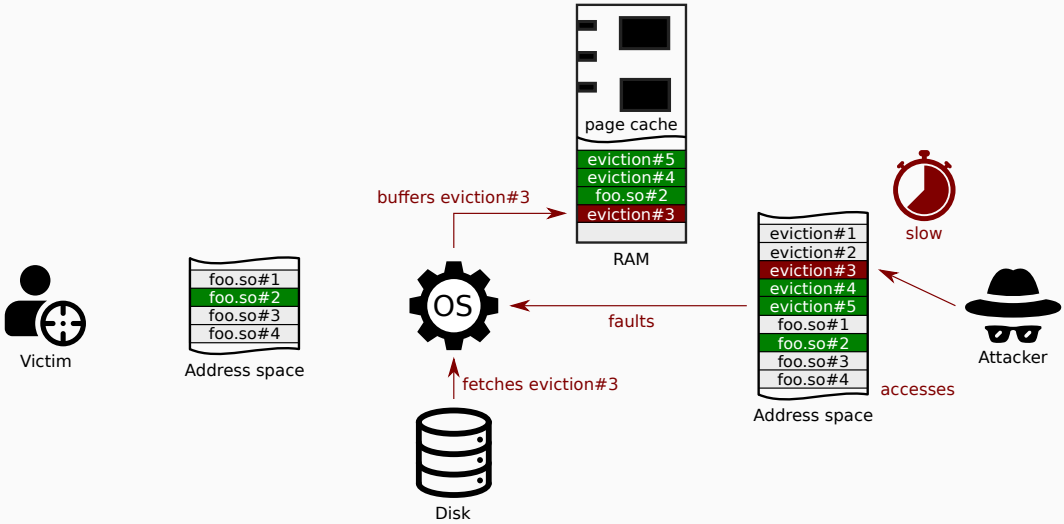
Page Cache Attacks

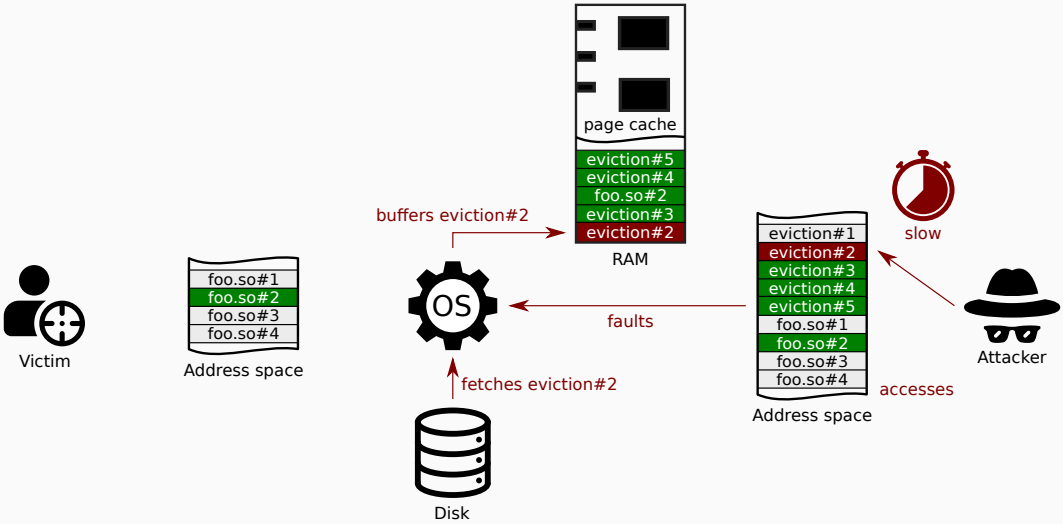


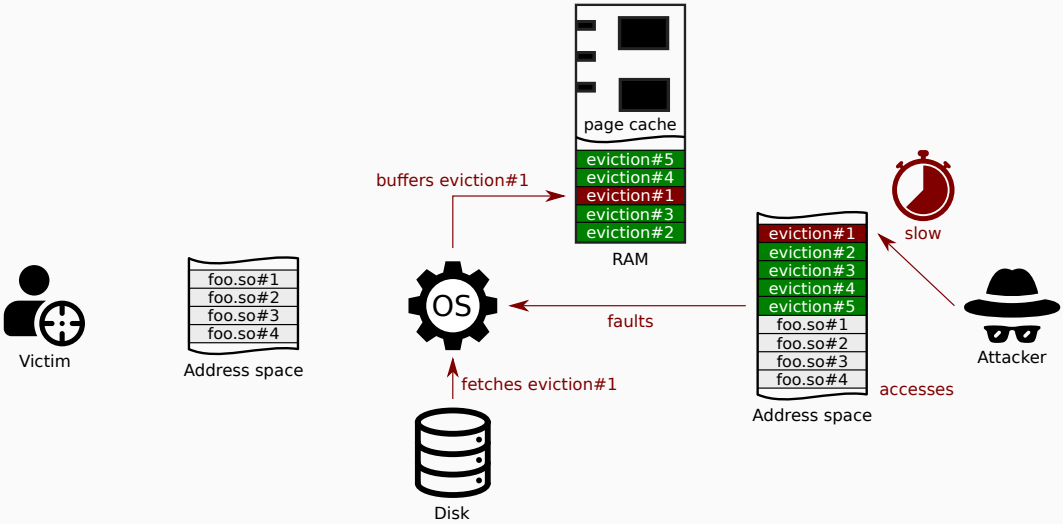
Page Cache Attacks



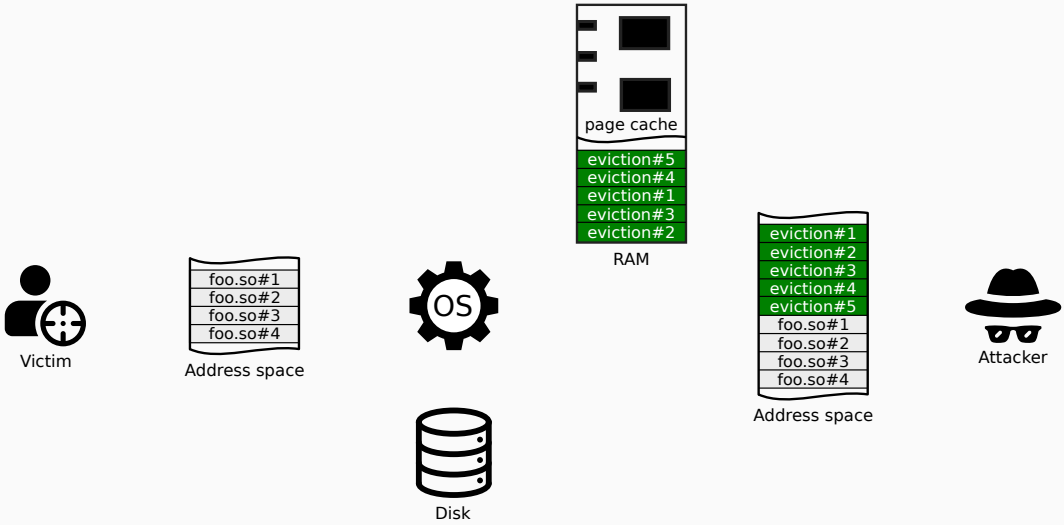

page cache
eviction#5
eviction#4
foo.so#2
RAM
buffers eviction#4
slow
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
OS
faults
Attacker
accesses
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
fetches eviction#4
Disk


page cache
eviction#5
eviction#4
foo.so#2
eviction#3
RAM
buffers eviction#3
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
OS
faults
Attacker
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
fetches eviction#3
accesses
Disk

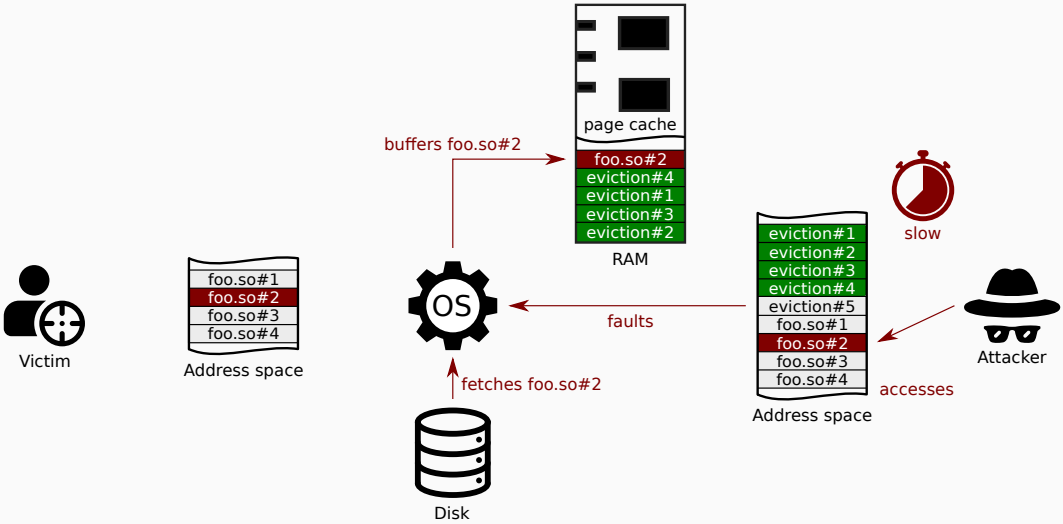

page cache
eviction#5
eviction#4
foo.so#2
eviction#3
eviction#2
RAM
buffers eviction#2
OS
faults
fetches eviction#2
Disk
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
slow
Attacker
accesses


page cache
eviction#5
eviction#4
eviction#1
eviction#3
eviction#2
RAM
buffers eviction#1
OS
faults
fetches eviction#1
Disk
Victim
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
eviction#1
eviction#2
eviction#3
eviction#4
eviction#5
foo.so#1
foo.so#2
foo.so#3
foo.so#4
Address space
slow
accesses
Attacker

Page Cache Attacks



Page Cache Attacks



`mincore` (2.04 μs)

`mincore` (2.04 μs)

- Takes virtual memory range, returns vector

`mincore` (2.04 μs)

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

`mincore` (2.04 μs)

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

`QueryWorkingSetEx` (465.91 ns)

`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

`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 ...

`mincore` (2.04 μs)

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

`QueryWorkingSetEx` (465.91 ns)

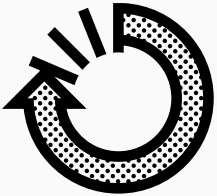
- Takes process handle + virtual memory address, returns struct
- Exposes attributes of queried page ...
- ... presence in working set

`mincore` (2.04 μs)

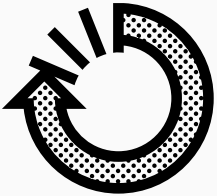
- Takes virtual memory range, returns vector
- Indicates presence of queried pages in page cache

`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`)

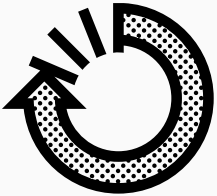


- Necessary for detecting multiple accesses

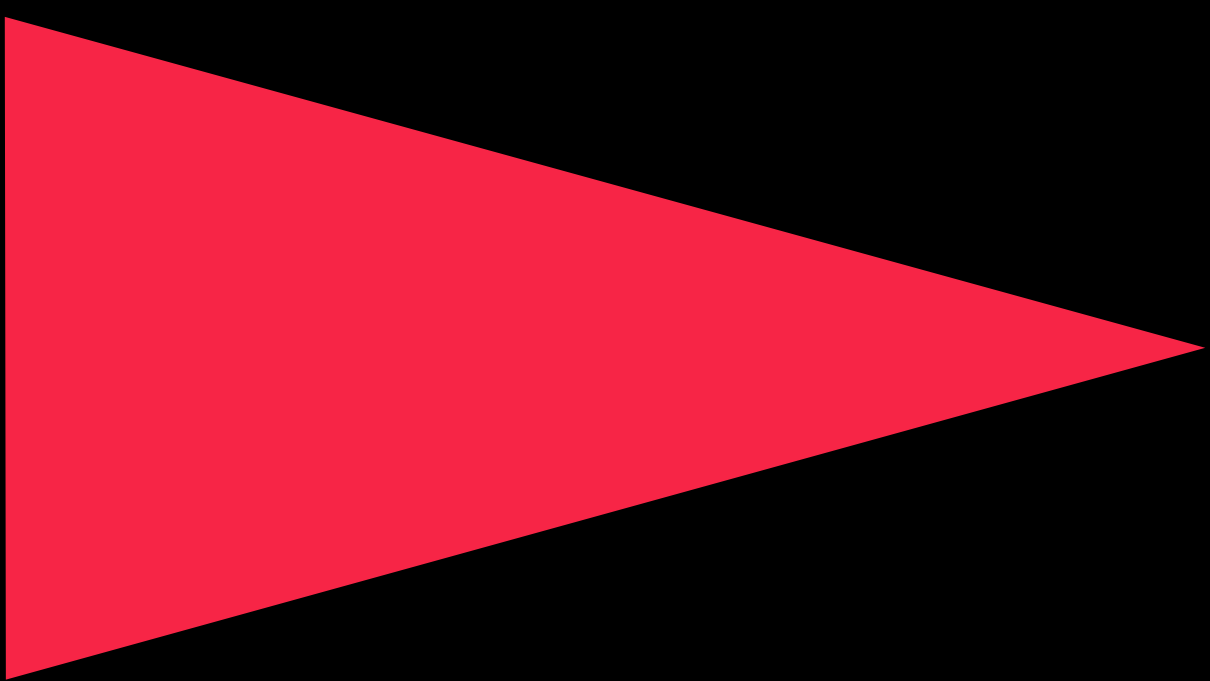


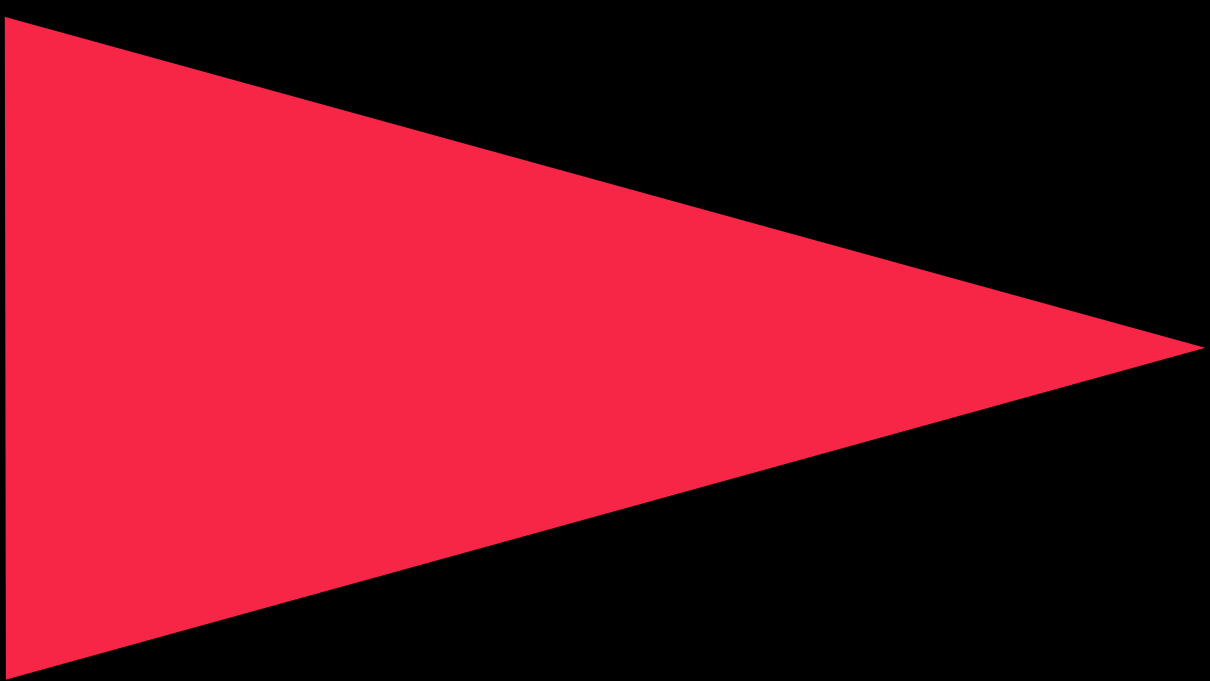
- Necessary for detecting multiple accesses
- Bottleneck of side channel

Reset Page Cache State



- Necessary for detecting multiple accesses
- Bottleneck of side channel
- Linux: eviction (takes 149 ms)
- Windows: `VirtualUnlock` if possible (takes 17.69 μs)
- Windows: `SetProcessWorkingSetSize` + eviction (takes 4.48 ms)





- 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

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

- 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 $\rightarrow$ more side channels
- But: low hanging fruits will disappear

Software-based Microarchitectural Attacks

Daniel Gruss

July 8, 2018

Graz University of Technology

References

D. Gruss, C. Maurice, and S. Mangard. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In: DIMVA. 2016.

D. Gruss, R. Spreitzer, and S. Mangard. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In: USENIX Security Symposium. 2015.

M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. ARMageddon: Cache Attacks on Mobile Devices. In: USENIX Security Symposium. 2016.

Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis. The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In: CCS. 2015.

M. Schwarz, D. Gruss, S. Weiser, C. Maurice, and S. Mangard. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In: DIMVA. 2017.