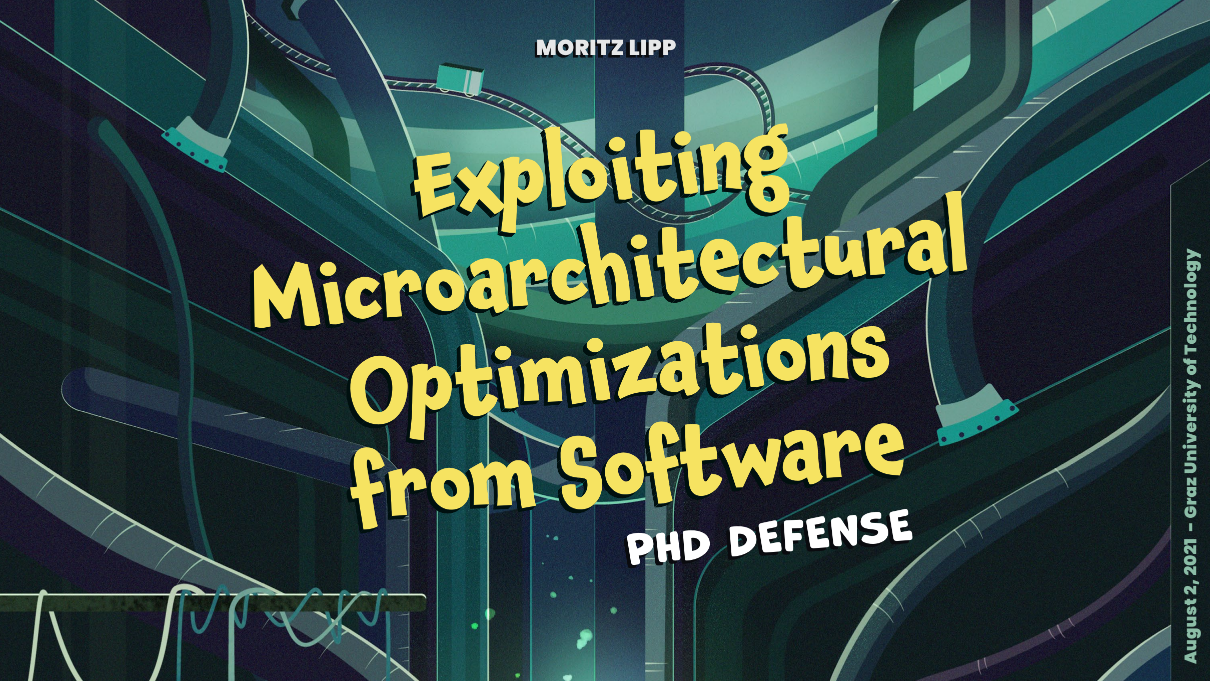


MORITZ LIPP

# Exploiting Microarchitectural Optimizations from Software

PHD DEFENSE

August 2, 2021 - Graz University of Technology

# Moritz

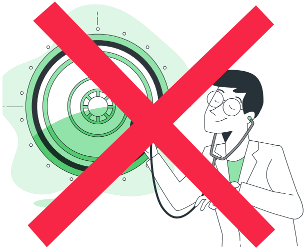
# Cooking

Cooking

Beekeeping

Cooking

Beekeeping



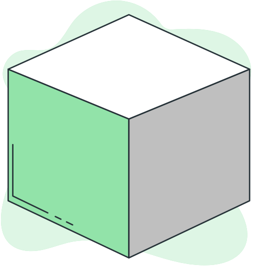
Side Channels

Side Channel Attacks are **Black Magic!**

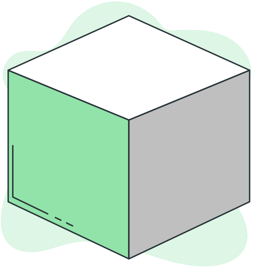
In my first semester as a student, . . .

. . . had to write a scientific report on side channel attacks.

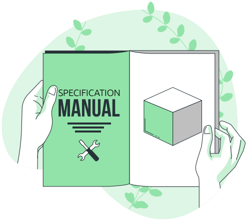
So, I did some research . . .



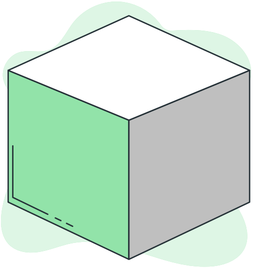
**Device**



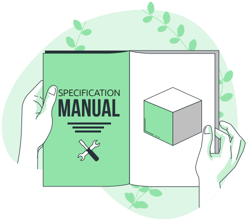
Device



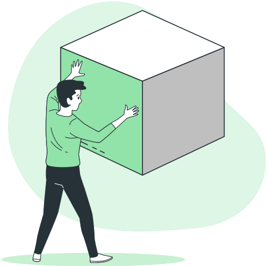
Specification



Device



Specification



Interaction

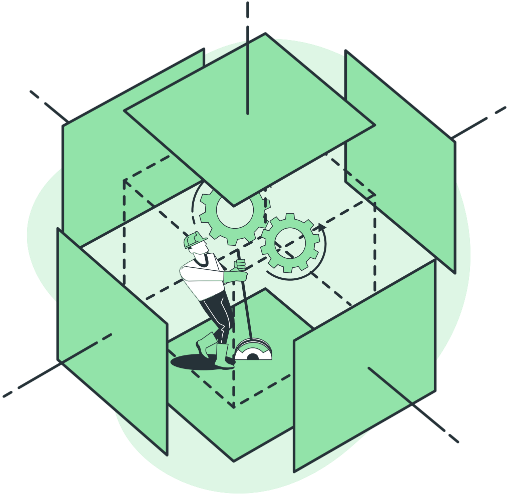
Everything works

as expected

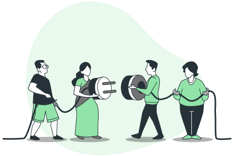
Everything works as expected

No bugs

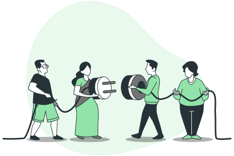
How can you attack this?



Information leaks through side effects

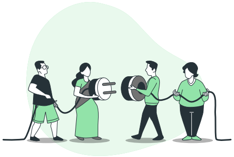


# Power Consumption



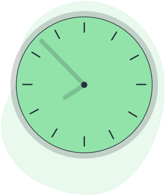
Power Consumption

Temperature

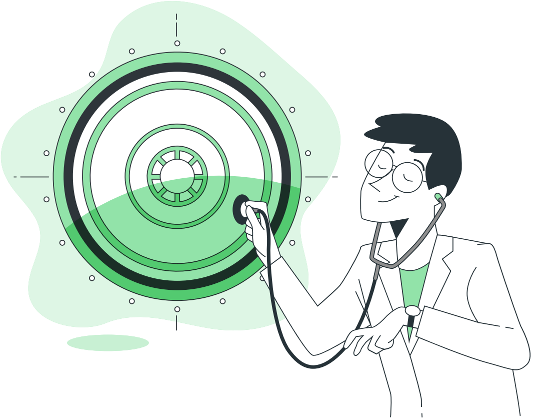


Power Consumption

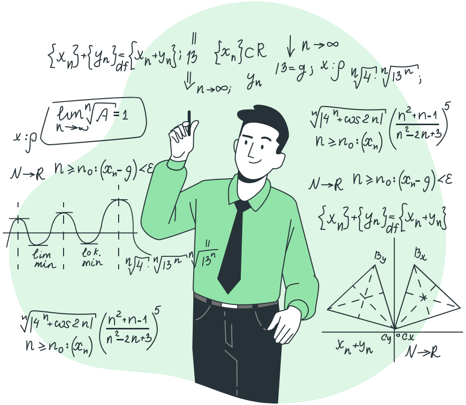
Temperature



Execution Time

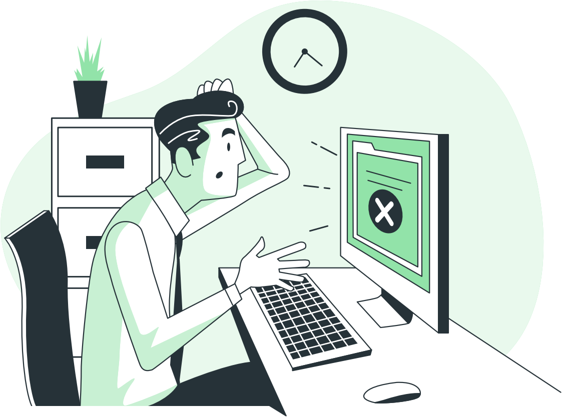


# Observe Device


$\{x_n\}+\{y_n\} \underset{df}{=} \{x_n+y_n\}$
$\{x_n\} \subset R$
$n \to \infty$
$\lim_{n\to\infty} \sqrt[n]{A} = 1$
$N \to R$
$n \geq n_0 : (x_n - g) < \varepsilon$
lim min
lok. min
$\sqrt[n]{4} : \sqrt[n]{13^n}$
$\left(\frac{n^2+n-1}{n^2-2n+3}\right)^5$
$n \geq n_0 : (x_n)$
$B_y$
$B_x$
$x_n + y_n$

# Evaluate data

Get the secrets. Easy as that!



**Witchcraft!** This is not for me!

Looking for a master thesis . . .

# Cryptography Tools

Cryptography Tools

Secure Code Generation

Cryptography Tools

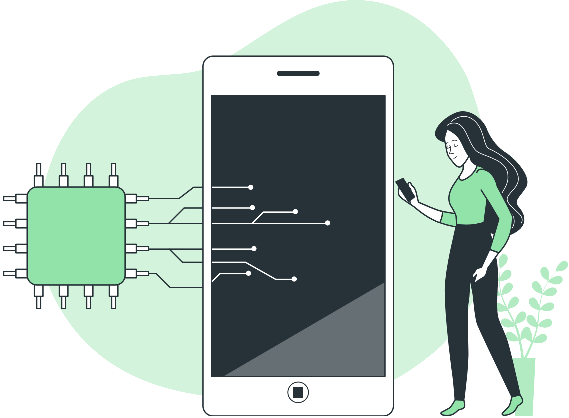
Secure Code Generation

Software-based Cache Attacks

Cryptography Tools

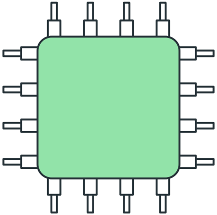
Compiler Extensions

**Software-based Cache Attacks**



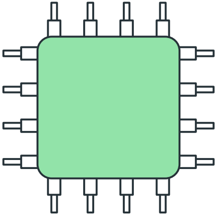
# Cache Attacks on ARM

# Architecture vs Microarchitecture



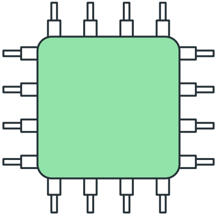
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . . )

# Architecture vs Microarchitecture



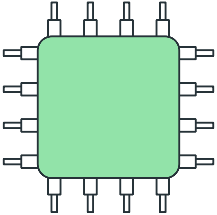
- Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, . . . )
- Interface between hardware and software

# Architecture vs 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 vs 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

AMD 64 Athlon™

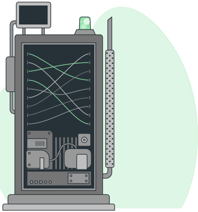
AMD RYZEN 5

intel® inside™ CORE™ i7

intel® inside™ Xeon®

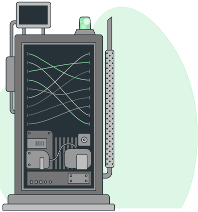
•••

# Optimizations



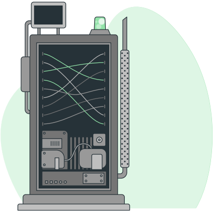
- Modern CPUs contain multiple microarchitectural elements

# Optimizations



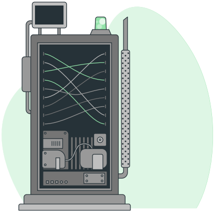
- Modern CPUs contain multiple microarchitectural elements
- Transparent for the programmer

# Optimizations



- Modern CPUs contain multiple microarchitectural elements
- Transparent for the programmer
- Optimize for performance, power consumption, . . .

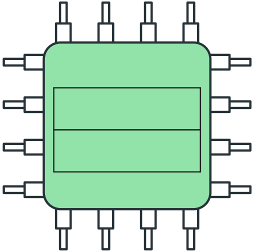
# Optimizations

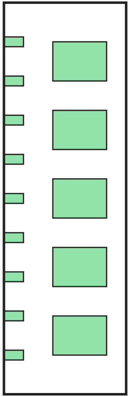


- Modern CPUs contain multiple microarchitectural elements
- Transparent for the programmer
- Optimize for performance, power consumption, . . .

# Cache

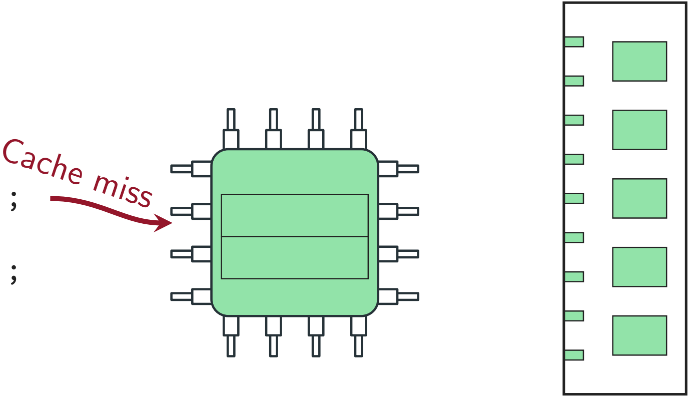
```
printf("%d", i);

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





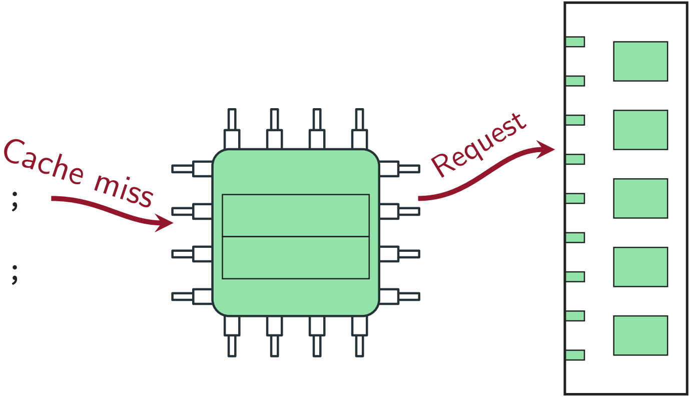
# Cache

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

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



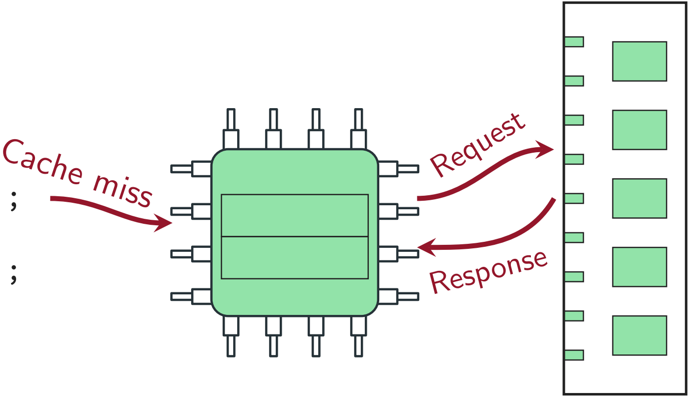
# Cache

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

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



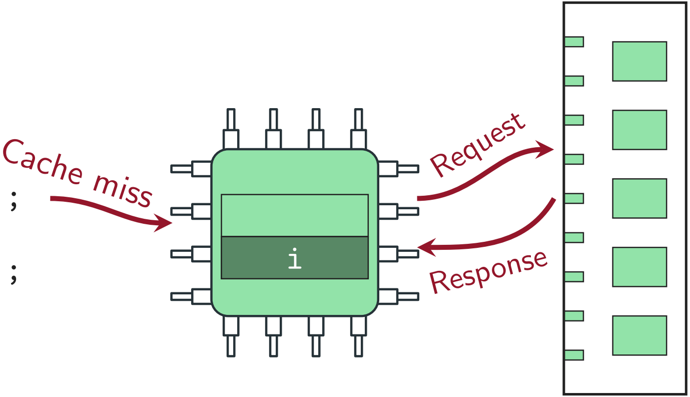
# Cache

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

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

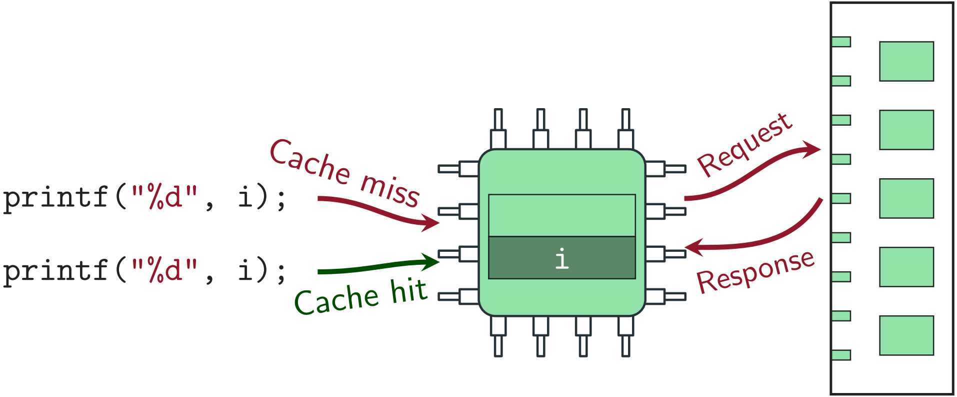


# Cache

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

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



# Cache



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

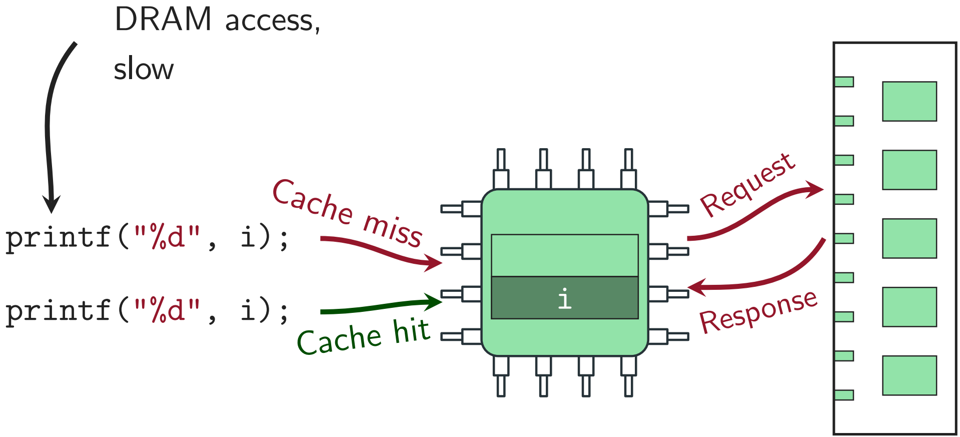
Cache miss

Cache hit

Request

Response

# Cache



DRAM access, slow

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

Cache miss

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

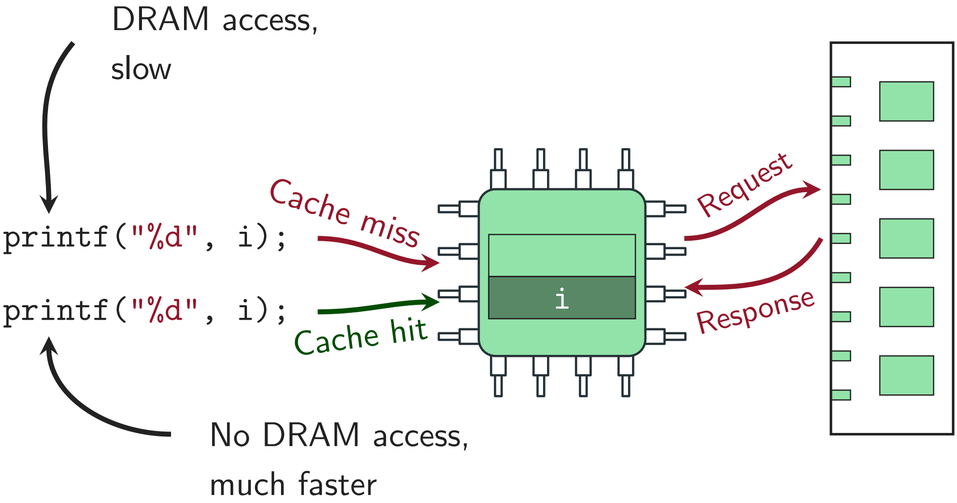
Cache hit

i

Request

Response

# Cache



DRAM access,
slow

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

Cache miss

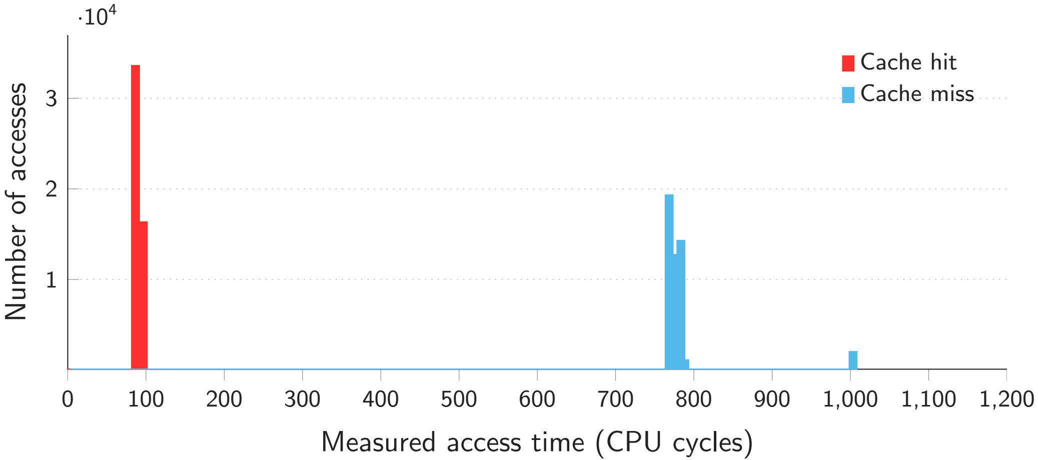
Cache hit

Request

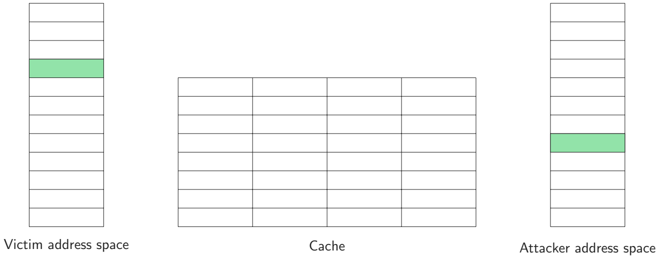
Response

No DRAM access,
much faster

# Caching speeds up Memory Accesses

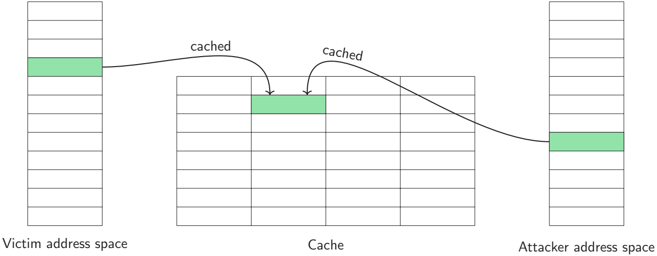


# Flush+Reload



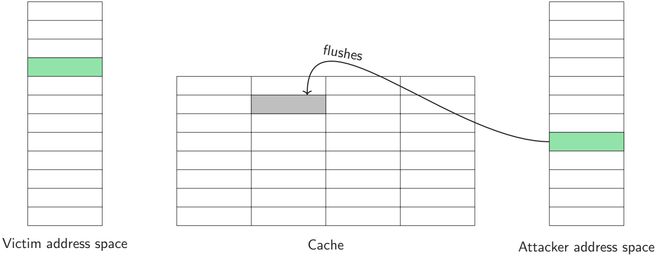
**Step 1:** Attacker maps shared library (shared memory, in cache)

# Flush+Reload



**Step 1:** Attacker maps shared library (shared memory, in cache)

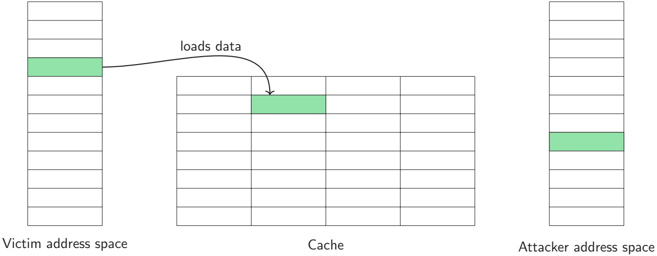
# Flush+Reload



**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker flushes the shared cache line

# Flush+Reload

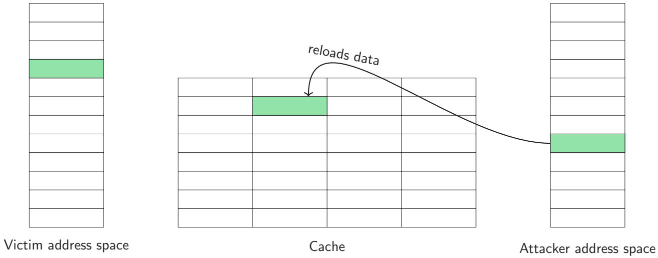


**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker flushes the shared cache line

**Step 3:** Victim loads the data

# Flush+Reload



**Step 1:** Attacker maps shared library (shared memory, in cache)

**Step 2:** Attacker flushes the shared cache line

**Step 3:** Victim loads the data

**Step 4:** Attacker measures the access time to reload the data

# Cache Attacks

- Leak cryptographic keys

# Cache Attacks

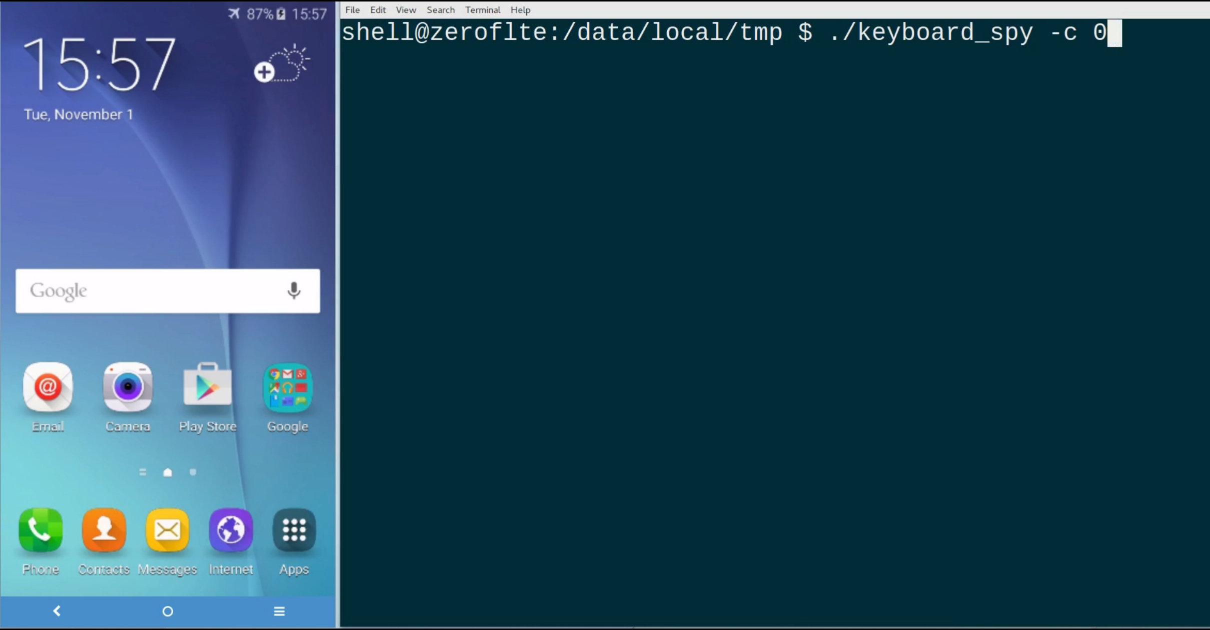
- Leak cryptographic keys
- Leak information on co-located virtual machines

# Cache Attacks

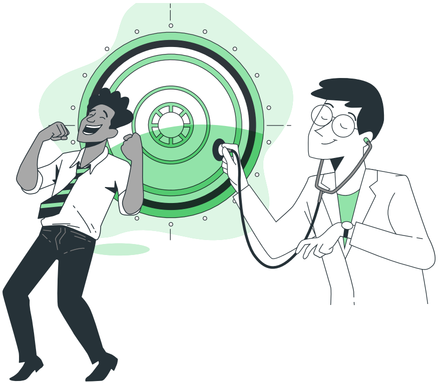
- Leak cryptographic keys
- Leak information on co-located virtual machines
- Monitor function calls of other applications

# Cache Attacks

- Leak cryptographic keys
- Leak information on co-located virtual machines
- Monitor function calls of other applications
- Build covert communication channels
- ...



```
shell@zeroflte:/data/local/tmp $ ./keyboard_spy -c 0
```

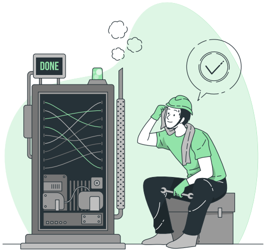


# Side-Channel Attacks are Fun

. . . and I started a PhD

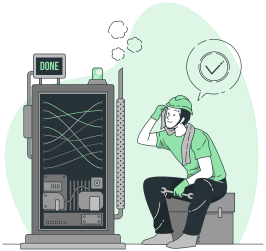
# What my research is about . . .

# Motivation



**Abstraction**

# Motivation



Abstraction

Optimizations

# Performance Optimizations

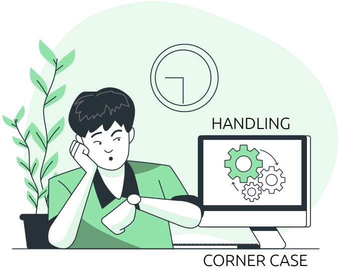
**Common Case**

Make it fast

# Performance Optimizations

**Common Case**

Make it fast



**Corner Case**

Make sure to handle it

# Motivation

**Understand Inner Workings**

# Motivation

**Understand Inner Workings**

**Security Implications**

# Properties

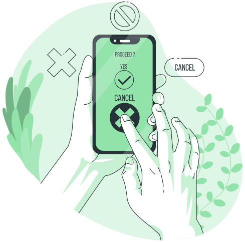
**Software-only**

No Physical Access required

# Properties

**Software-only**

No Physical Access required



**Misuse Interfaces**

Trigger Corner Cases

# Contribution

*Advance* the state of the art of *microarchitectural attacks and defenses*.

- Discovering **transient-execution attacks**.

# Contribution

*Advance* the state of the art of *microarchitectural attacks and defenses*.

- Discovering **transient-execution attacks**.
- Identify previously **unknown attack vectors**.

# Contribution

*Advance* the state of the art of *microarchitectural attacks and defenses*.

- Discovering **transient-execution attacks**.
- Identify previously **unknown attack vectors**.
- Exploring if **different existing attacks** can be mounted remotely.

# Contribution

*Advance* the state of the art of *microarchitectural attacks and defenses*.

- Discovering **transient-execution attacks**.
- Identify previously **unknown attack vectors**.
- Exploring if **different existing attacks** can be mounted remotely.
- Combining traditional **physical side-channel analysis** with modern **software-based** microarchitectural attack techniques.

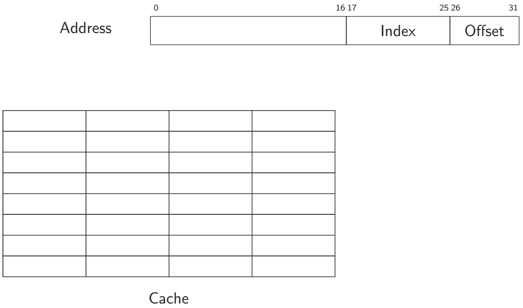
# Contribution

*Advance* the state of the art of *microarchitectural attacks and defenses*.

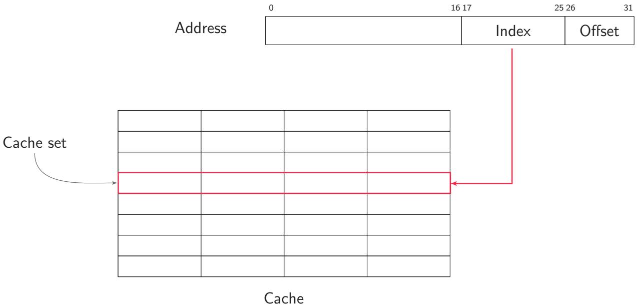
- Discovering **transient-execution attacks**.
- Identify previously **unknown attack vectors**.
- Exploring if **different existing attacks** can be mounted remotely.
- Combining traditional **physical side-channel analysis** with modern **software-based** microarchitectural attack techniques.
- Giving **new insights** into efficiently mitigating attacks.

# Way Prediction

# Set-Associative Caches

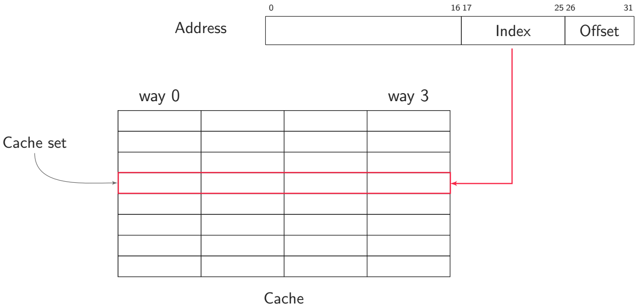


# Set-Associative Caches



Data loaded in a specific set depending on its address

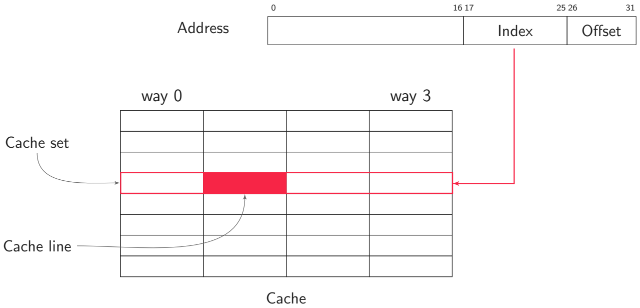
# Set-Associative Caches



Data loaded in a specific set depending on its address

Several ways per set

# Set-Associative Caches



Data loaded in a specific set depending on its address

Several ways per set

# Way Prediction

- In a set-associative cache, bits in the address determine in which set the cache line is located.

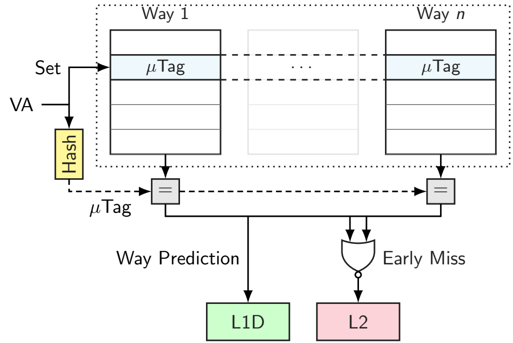
# Way Prediction

- In a set-associative cache, bits in the address determine in which set the cache line is located.
- With an $n$-way cache, $n$ possible entries need to be checked.

# Way Prediction

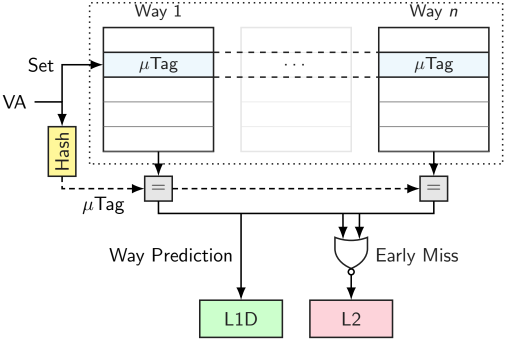
- In a set-associative cache, bits in the address determine in which set the cache line is located.
- With an *n*-way cache, *n* possible entries need to be checked.
- Using **way prediction** [4], one entry is predicted
  - Correct prediction: Access completed
  - Incorrect prediction: Perform associate check

# AMD Way Predictor



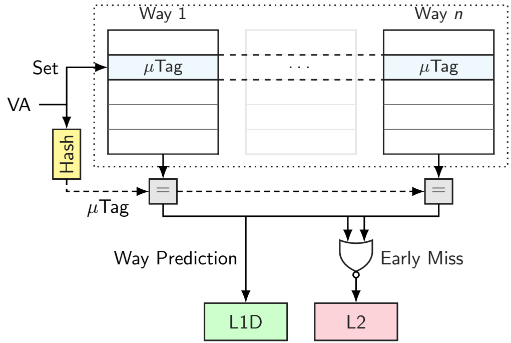
- Introduced with the AMD Bulldozer microarchitecture

# AMD Way Predictor



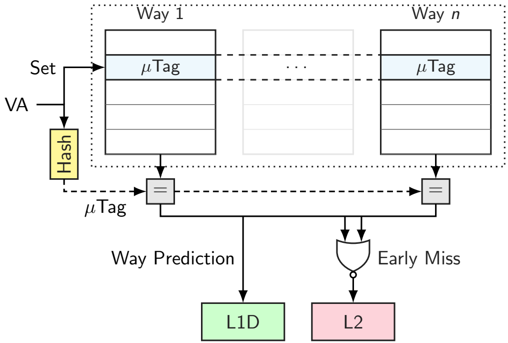
- Introduced with the AMD Bulldozer microarchitecture
- Every cache line in the L1D is tagged with a $\mu$Tag

# AMD Way Predictor



- Introduced with the AMD Bulldozer microarchitecture
- Every cache line in the L1D is tagged with a $\mu$Tag
- Predicts the cache way based on this $\mu$Tag
    - Saving power and reduces bank conflicts

# AMD Way Predictor

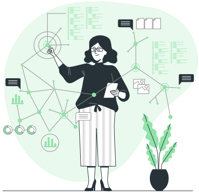


- Introduced with the AMD Bulldozer microarchitecture
- Every cache line in the L1D is tagged with a $\mu$Tag
- Predicts the cache way based on this $\mu$Tag
  - Saving power and reduces bank conflicts
- No match for $\mu$Tag, detect early miss and issue L2 request

# AMD Way Predictor

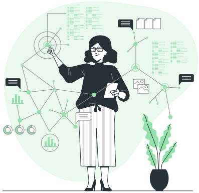
- Two different virtual addresses with the same $\mu$Tag but different physical addresses will conflict

# Hash Function



- L1D way predictor computes a hash ($\mu$Tag) from the virtual address

# Hash Function



- L1D way predictor computes a hash ($\mu$Tag) from the virtual address
- This hash function is **not documented**

# Recovering the Hash Function

- Rely on $\mu$Tag collisions to reverse-engineer the hash function

## Recovering the Hash Function

- Rely on $\mu$Tag collisions to reverse-engineer the hash function
- Pick two random virtual addresses mapping to the same cache set

# Recovering the Hash Function

- Rely on $\mu$Tag collisions to reverse-engineer the hash function
- Pick two random virtual addresses mapping to the same cache set
- Access them repeatedly

# Recovering the Hash Function

- Rely on $\mu$Tag collisions to reverse-engineer the hash function
- Pick two random virtual addresses mapping to the same cache set
- Access them repeatedly
- If they have the **same $\mu$Tag**:
  - Increased access time
  - Increased number of performance counter for L1 misses

# Recovering the Hash Function

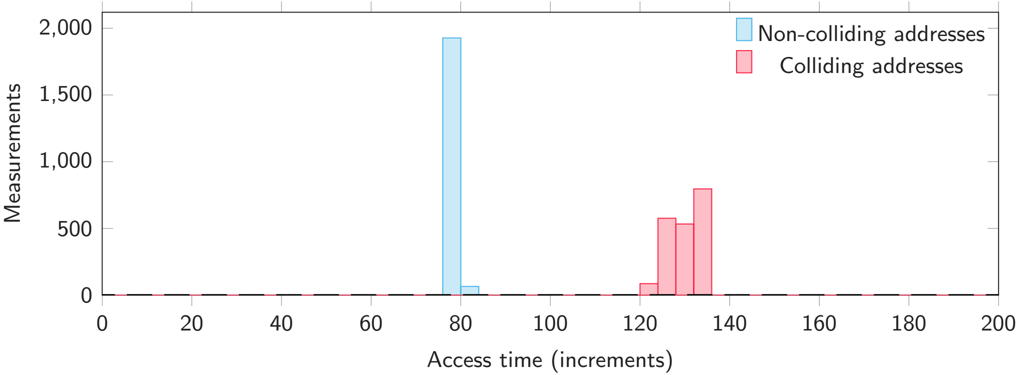
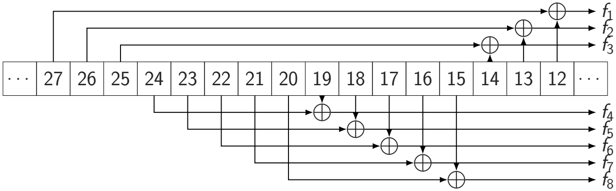
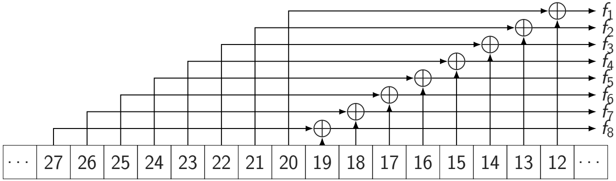


**Figure 1:** Measured duration of 250 alternating accesses to addresses with and without the same $\mu$Tag.

# Recovering the Hash Function

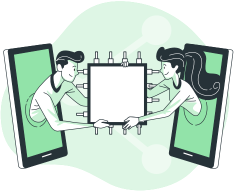


**(a)** Zen, Zen+, Zen 2



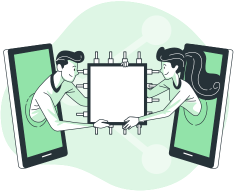
**(b)** Bulldozer, Piledriver, Steamroller

# Case Studies



Covert Channel

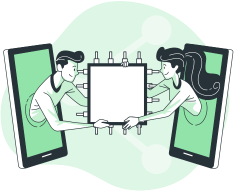
# Case Studies



Covert Channel

Break AES

# Case Studies



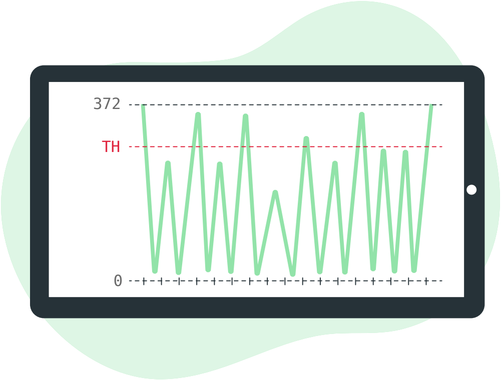
Covert Channel

Break AES

Break KASLR

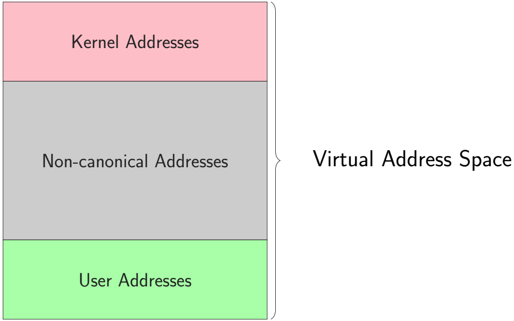
# Publication

**Lipp, M.**, Hadžić, V., Schwarz, M., Perais, A., Maurice, C., Gruss, D., “Take a Way: Exploring the Security Implications of AMD’s Cache Way Predictors”. In: *AsiaCCS*. 2020



Always leaking **metadata** . . .

# Virtual Memory



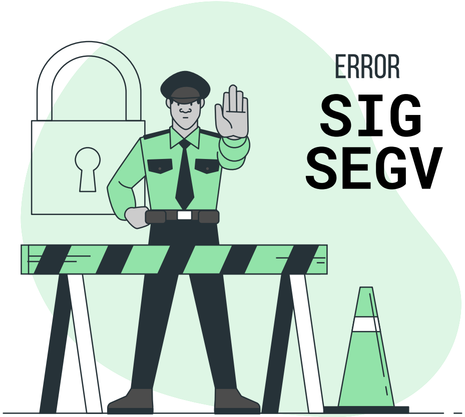
# Building the Code

- Find something human readable, e.g., the Linux version

```
# sudo grep linux_banner /proc/kallsyms
ffffffff81a000e0 R linux_banner
```

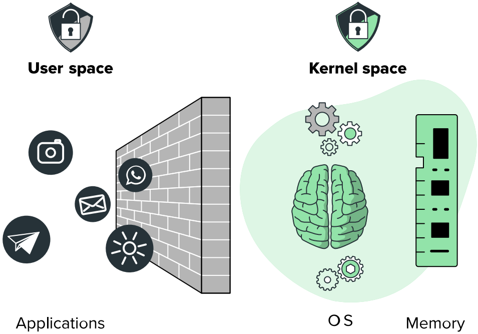
# Building the Code

```
char data = *(char*) 0xffffffff81a000e0;
printf("%c\n", data);
```



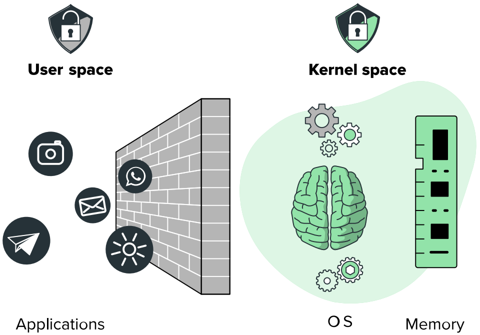
Invalid Access throws an Exception

# Memory Isolation



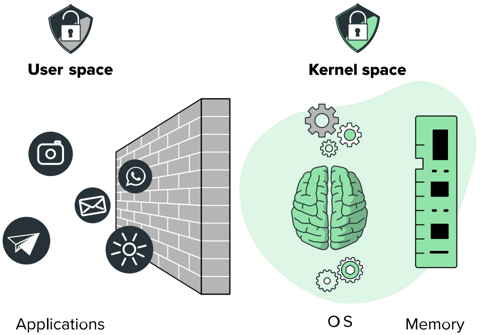
- Kernel is isolated from user space

# Memory Isolation



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

# Memory Isolation



- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel

# Paging

- CPU support virtual address spaces to isolate processes

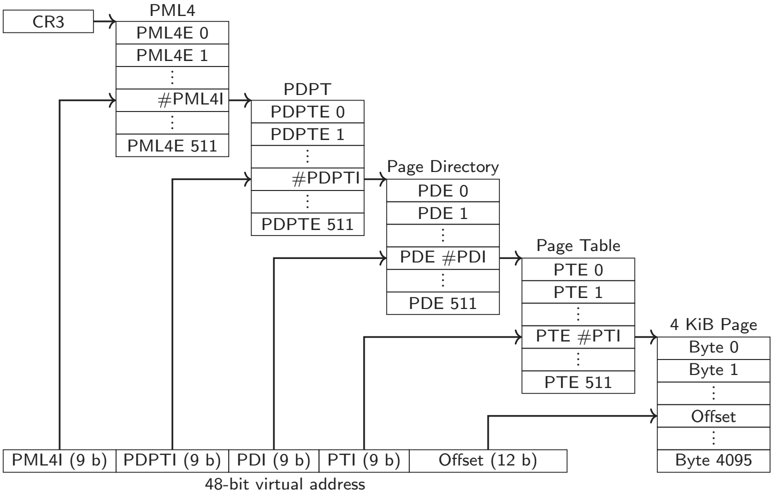
# Paging

- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames

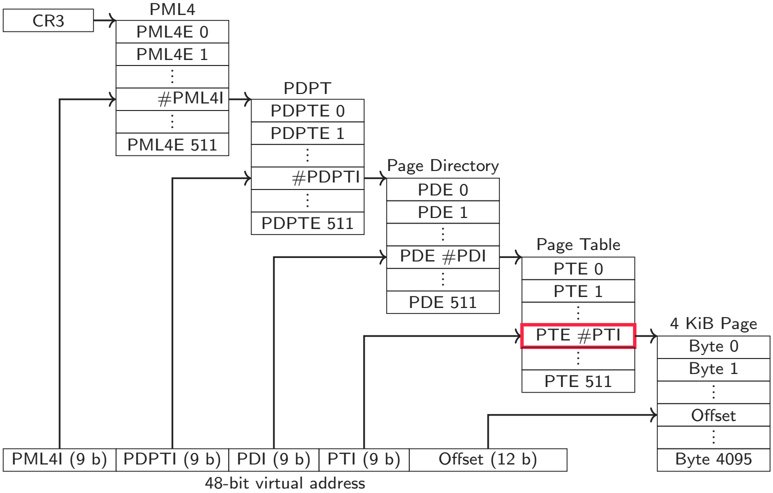
# Paging

- CPU support virtual address spaces to isolate processes
- Physical memory is organized in page frames
- Virtual memory pages are mapped to page frames using page tables

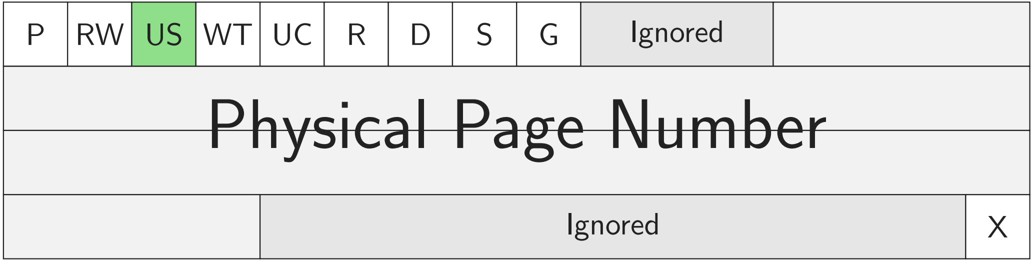
# Address Translation on x86-64



# Address Translation on x86-64



# Page Table Entry



P RW US WT UC R D S G Ignored

Physical Page Number

Ignored X

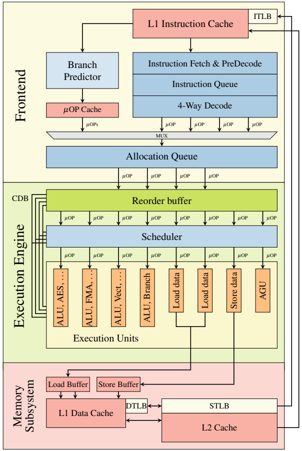
- User/Supervisor bit defines in which privilege level the page can be accessed

# Loading an address

```c
char data = *(char*) 0xffffffff81a000e0;
printf("%c\n", data);
```

- We try to load an inaccessible address
- Permission is checked

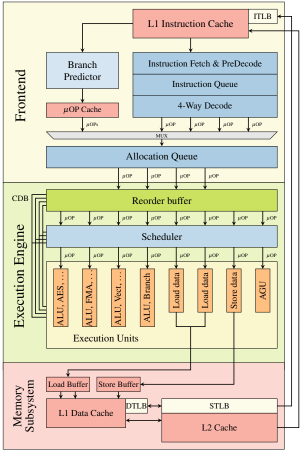
# Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end

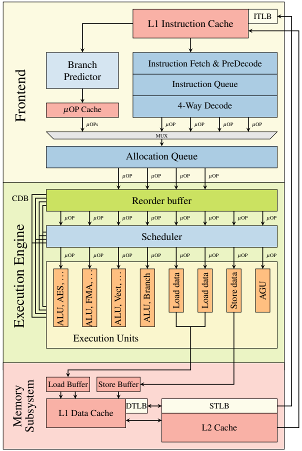
# Out-of-Order Execution



Instructions are

- fetched and decoded in the front-end
- dispatched to the backend

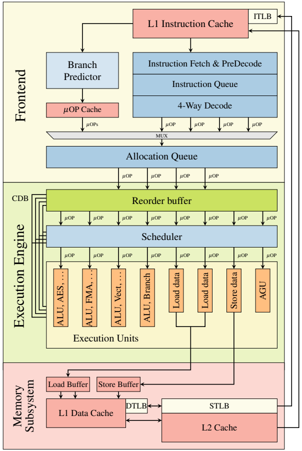
# Out-of-Order Execution



Instructions are

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

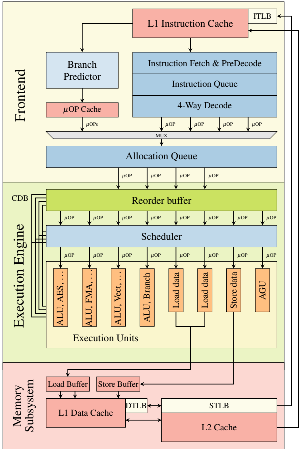
# Out-of-Order Execution



## Instructions

- are executed out-of-order

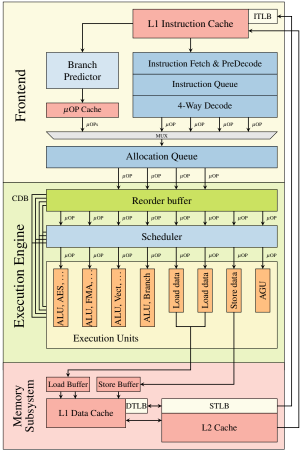
# Out-of-Order Execution



Instructions

- are executed out-of-order
- wait until their dependencies are ready

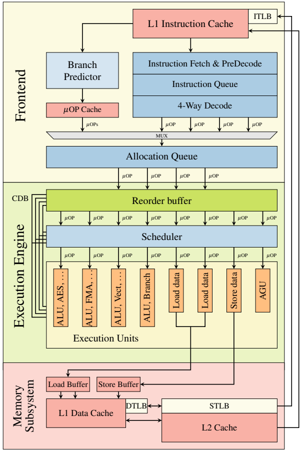
# Out-of-Order Execution



Instructions

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

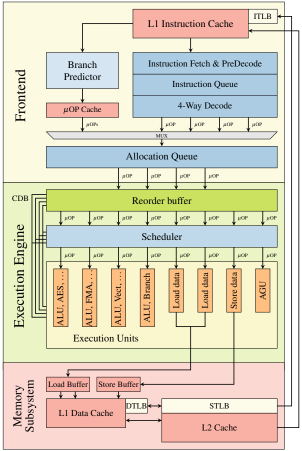
# Out-of-Order Execution



Instructions

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

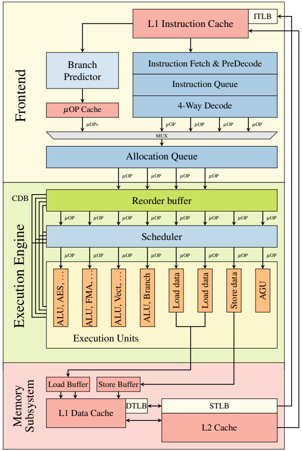
# Out-of-Order Execution



Instructions

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

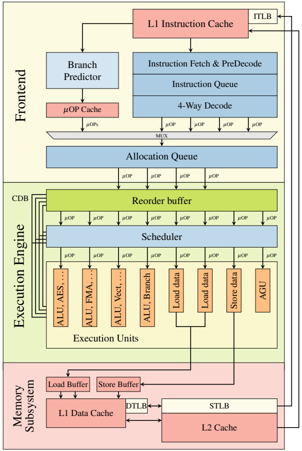
# Out-of-Order Execution



Instructions

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

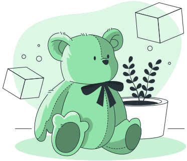
# Out-of-Order Execution



Instructions

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

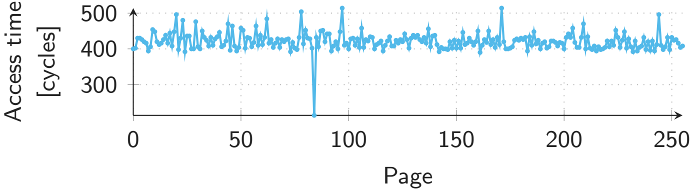
# Toy example



```
*(volatile char*) 0; // raise_exception();
array[84 * 4096] = 0;
```

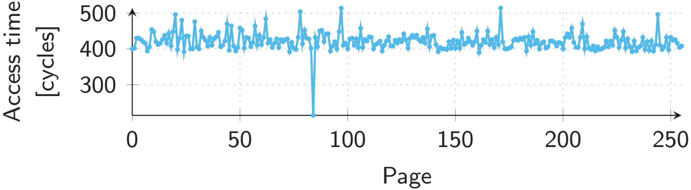
# Building the Code

- Flush+Reload over all pages of the array



# Building the Code

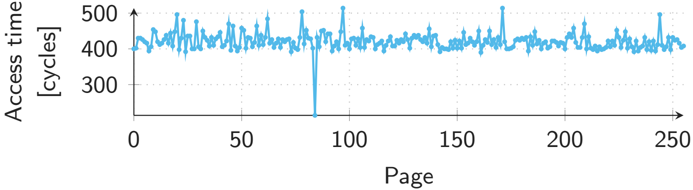
- Flush+Reload over all pages of the array



- “Unreachable” code line was actually executed

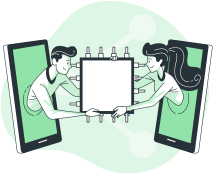
# Building the Code

- Flush+Reload over all pages of the array



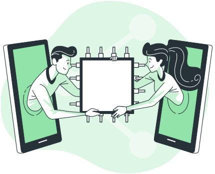
- “Unreachable” code line was actually executed
- Exception was only thrown afterwards

# Building a Covert Channel



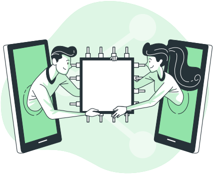
- Transfer of the microarchitectural state into an architectural state

# Building a Covert Channel



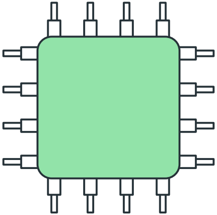
- Transfer of the microarchitectural state into an architectural state
- Transient instruction sequence is the sender

# Building a Covert Channel



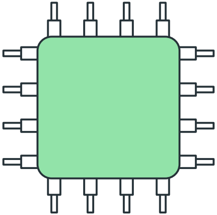
- Transfer of the microarchitectural state into an architectural state
- Transient instruction sequence is the sender
- Receiver receives the microarchitectural state change and deduces the secret from the state

# Building a Covert Channel



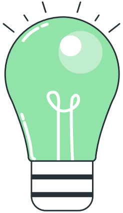
- Leverage techniques from cache attacks: Flush+Reload
- Transmit multiple bits at once
  - 256 different byte values ⇒ access different cache line

# Building a Covert Channel



- Leverage techniques from cache attacks: Flush+Reload
- Transmit multiple bits at once
  - 256 different byte values ⇒ access different cache line
- Not limited to the cache

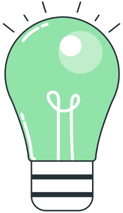
# Building the Code



- Add another layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

# Building the Code



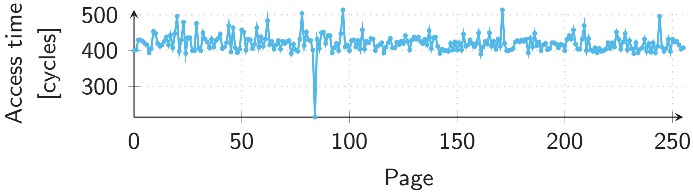
- Add another layer of indirection to test

```
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

- Then check whether any part of `array` is cached

# Building the Code

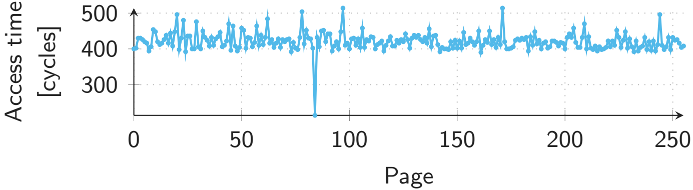
- Flush+Reload over all pages of the array



- Index of cache hit reveals data

# Building the Code

- Flush+Reload over all pages of the array



- Index of cache hit reveals data
- Permission check is in some cases not fast enough

# Meltdown

- Using out-of-order execution, we can read data at any address

# Meltdown

- Using out-of-order execution, we can read data at any address
- Entire physical memory is typically accessible through kernel space

# Meltdown

- Using out-of-order execution, we can read data at any address
- Entire physical memory is typically accessible through kernel space
- Bypass the most fundamental security guarantees

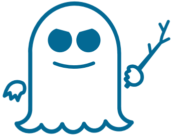
# Meltdown

- Using out-of-order execution, we can read data at any address
- Entire physical memory is typically accessible through kernel space
- Bypass the most fundamental security guarantees
- Can leak data directly, not only meta data

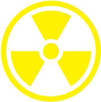
# Transient-Execution Attacks

With transient-execution attacks, a new research field emerged

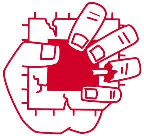
Meltdown



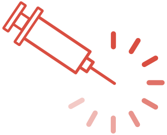
Spectre



Fallout



Zombieload



LVI

Medusa

# Publication

**Lipp, M.**, Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M., "Meltdown: Reading Kernel Memory from User Space". In: *USENIX Security Symposium*. 2018

# Operating System Microarchitecture

# KASLR: Kernel Address Space Layout Randomization

- Many exploits rely on the knowledge of the memory location of a certain function

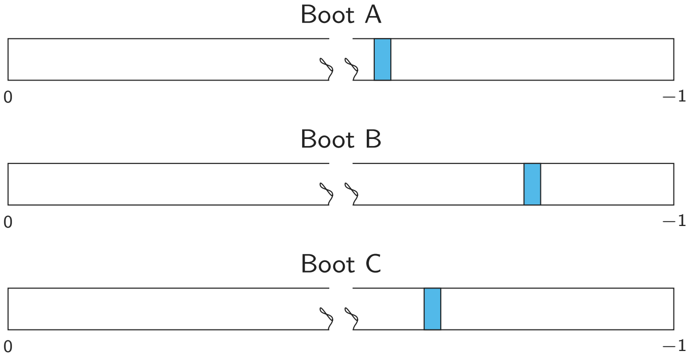
# KASLR: Kernel Address Space Layout Randomization

- Many exploits rely on the knowledge of the memory location of a certain function
- Statistical mitigation of memory corruption vulnerabilities

# KASLR: Kernel Address Space Layout Randomization

- Many exploits rely on the knowledge of the memory location of a certain function
- Statistical mitigation of memory corruption vulnerabilities
- Randomizing core kernel image and device drivers position at boot time

# KASLR: Kernel Address Space Layout Randomization

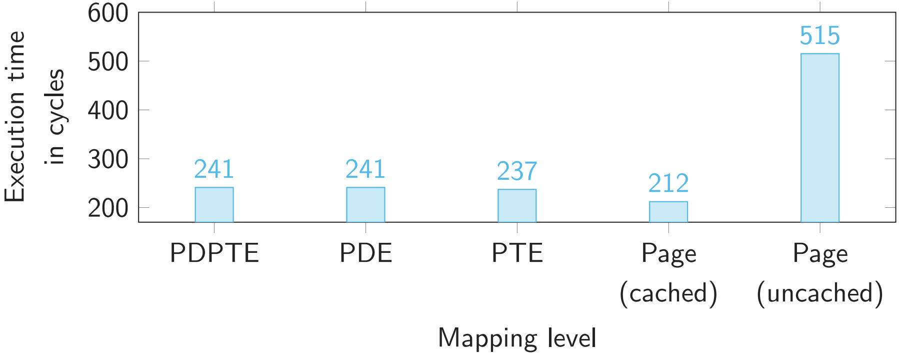


- Driver is loaded to a different offset on every boot

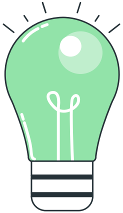
# Attacks

- Double Page Fault Attack [3]
  - Measuring execution time of page fault handler
- TSX Attack [5]
  - Measuring execution time of TSX abort handler
- Prefetch [2]
  - Execution time of prefetch instruction

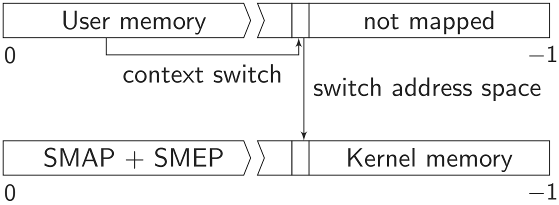
# Prefetch Side-Channel Attack



# Idea



- Stronger Kernel Address Isolation: Separate kernel space and user space



# Idea

- Every process has two address spaces:

# Idea

- Every process has two address spaces:
    - **Kernel Address Space**: Kernel mapped, user space mapped and protected with SMAP and SMEP

# Idea

- Every process has two address spaces:
  - **Kernel Address Space**: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - **Shadow Address Space**: User space mapped, Kernel not mapped

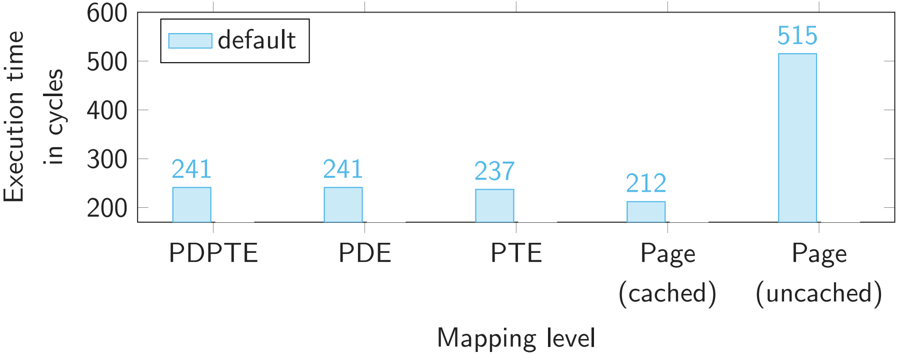
# Idea

- Every process has two address spaces:
  - **Kernel Address Space**: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - **Shadow Address Space**: User space mapped, Kernel not mapped
- Switching between the address space:

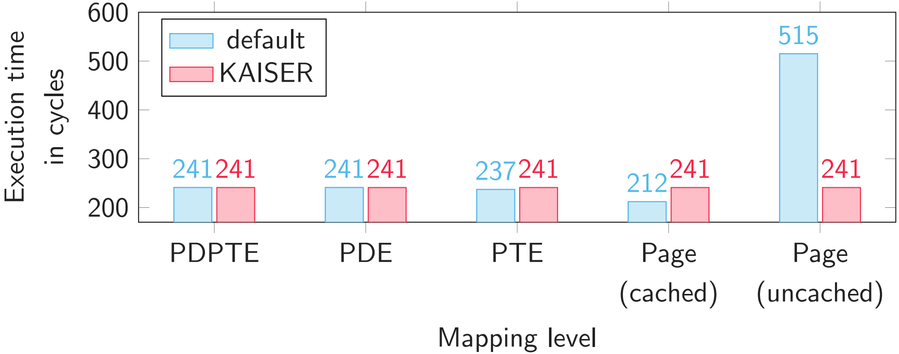
# Idea

- Every process has two address spaces:
  - **Kernel Address Space**: Kernel mapped, user space mapped and protected with SMAP and SMEP
  - **Shadow Address Space**: User space mapped, Kernel not mapped
- Switching between the address space:
  - Update CR3 with corresponding PML4

# Prefetch Side-Channel Attack



# Prefetch Side-Channel Attack



# Bonus

- For Meltdown, kernel addresses in user space are a problem

# Bonus

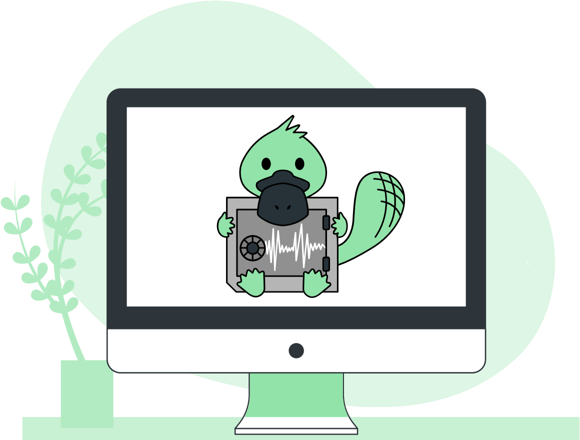
- For Meltdown, kernel addresses in user space are a problem
- With KAISER, these mappings are gone

# Bonus

- For Meltdown, kernel addresses in user space are a problem
- With KAISER, these mappings are gone
- Inadvertently defeats Meltdown as well
    - Incorporated to Linux, Apple and Windows

# Publication

Gruss, D., **Lipp, M.**, Schwarz, M., Fellner, R., Maurice, C., Mangard, S., “KASLR is Dead: Long Live KASLR”. In: *ESSoS*. 2017



# Software-based Power Side Channel Attacks

# Intel RAPL

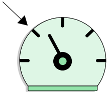
- Need for Platform Thermal Management, Platform Power Limiting, Power/Performance Budgeting

# Intel RAPL

- Need for Platform Thermal Management, Platform Power Limiting, Power/Performance Budgeting
- **Intel Running Average Power Limit** (RAPL) provides . . .

# Intel RAPL

- Need for Platform Thermal Management, Platform Power Limiting, Power/Performance Budgeting
- **Intel Running Average Power Limit** (RAPL) provides . . .



Power Limiting

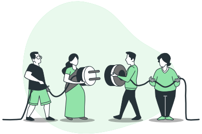
# Intel RAPL

- Need for Platform Thermal Management, Platform Power Limiting, Power/Performance Budgeting
- **Intel Running Average Power Limit** (RAPL) provides . . .

Power Limiting

Accurate Energy Reading

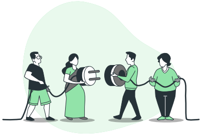
# Intel RAPL



- On **Linux**, counters can be accessed using the `powercap` framework

  `/sys/devices/virtual/powercap/intel-rapl`

# Intel RAPL



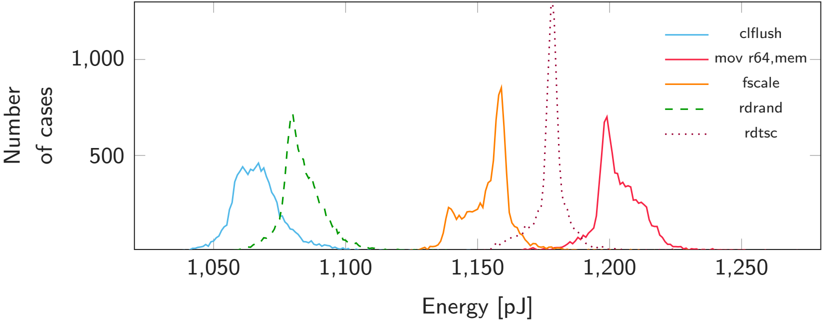
- On **Linux**, counters can be accessed using the `powercap` framework

  `/sys/devices/virtual/powercap/intel-rapl`

- On **macOS** and **Windows**, a driver from Intel needs to be installed

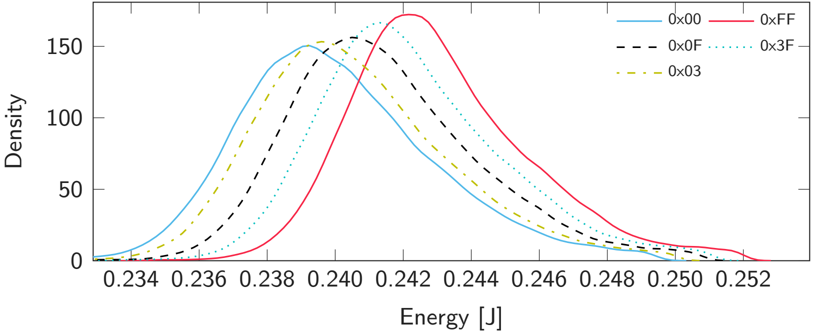
# Distinguishing Instructions

- Measure the energy consumption of **different instructions**



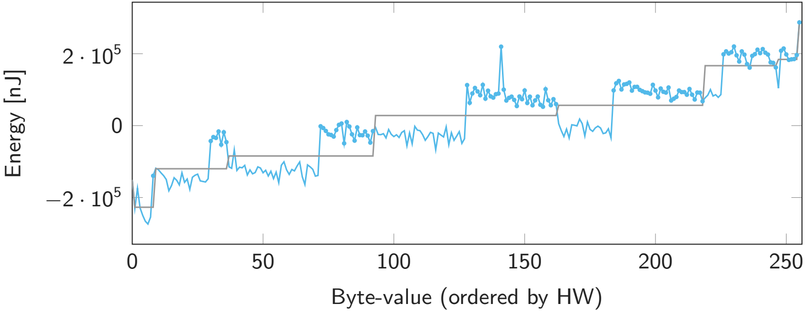
# Distinguishing Operands

- Measure the energy consumption of **different operands**

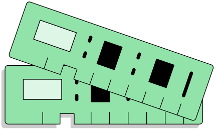


# Distinguishing Data

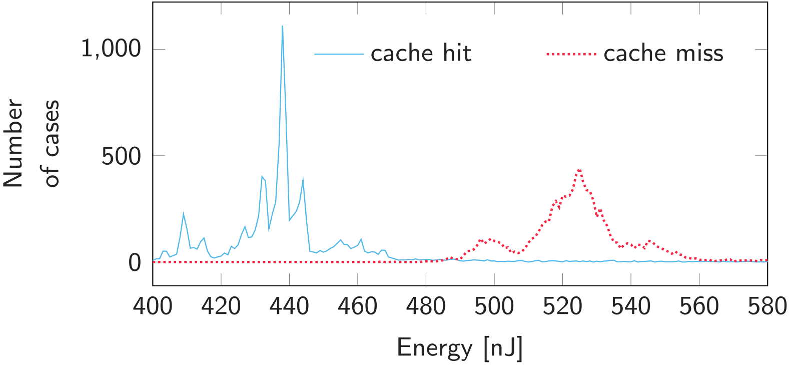
- Measure the energy consumption of **different load values**



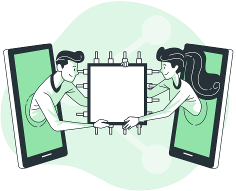
# Distinguishing Load Targets



- Measure the energy consumption of **different load targets**

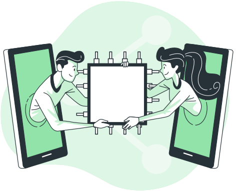


# Case Studies



Covert Channel

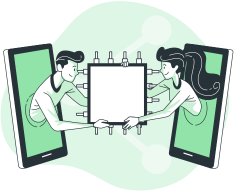
# Case Studies



Covert Channel

Break AES-NI

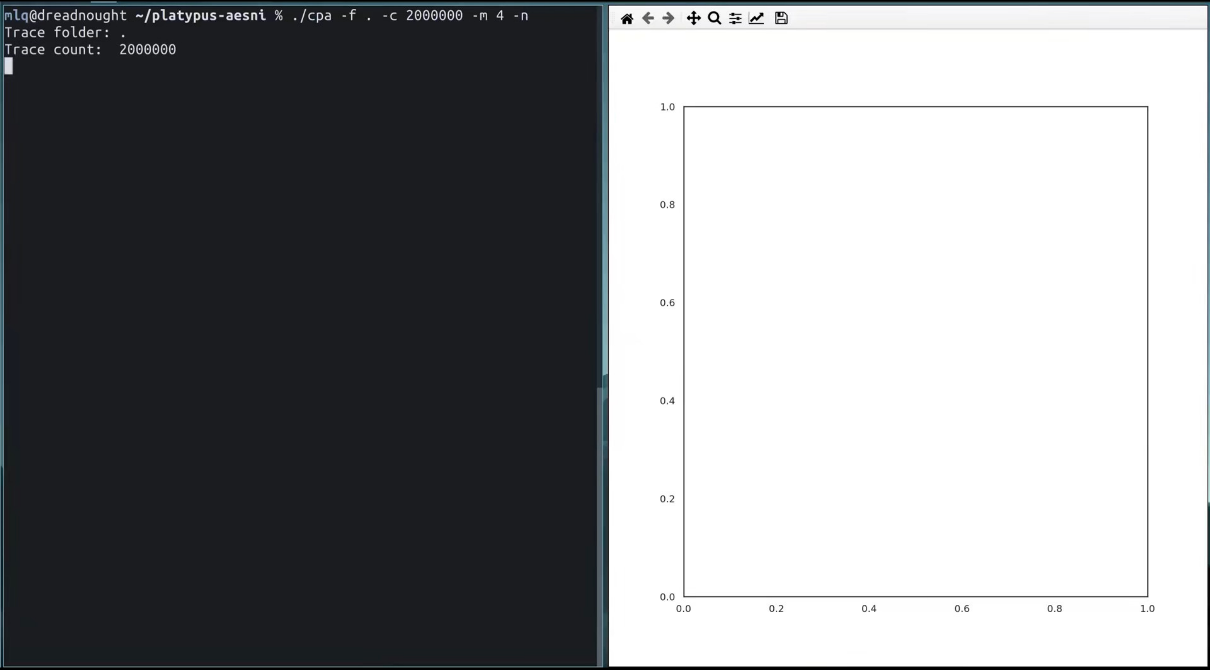
# Case Studies



Covert Channel

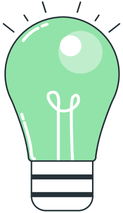
Break AES-NI

Break KASLR



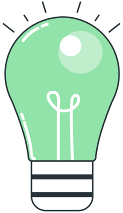
```
mlq@dreadnought ~/platypus-aesni % ./cpa -f . -c 2000000 -m 4 -n
Trace folder: .
Trace count:  2000000
```

# SGX Step



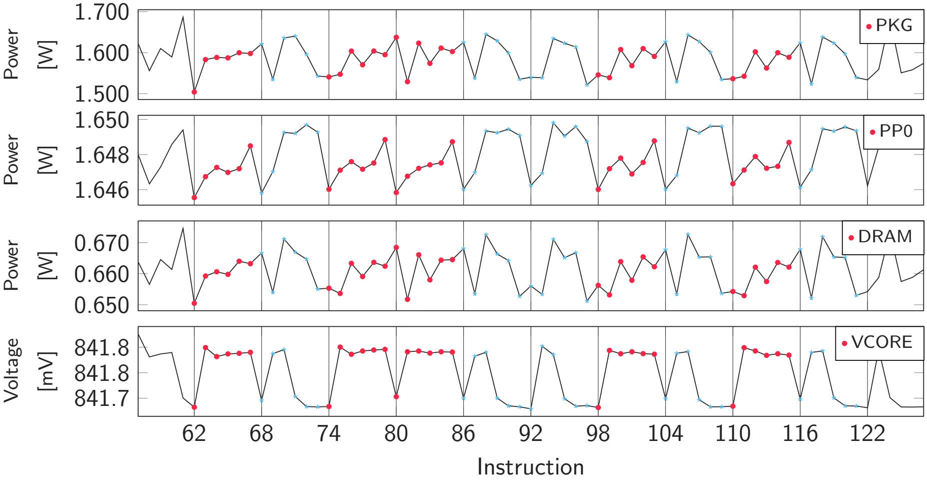
- **Combine** Intel RAPL with SGX-step

# SGX Step



- **Combine** Intel RAPL with SGX-step
- Measure the energy consumption of **single instructions**

# RSA Toy Cipher



# Publication

**Lipp, M.**, Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D., "PLATYPUS: Software-based Power Side-Channel Attacks on x86". In: *IEEE S&P*. 2021

# Interrupt-based Side Channel from JavaScript

# Motivation

- Acquire accurate timestamps of keystrokes for input sequences

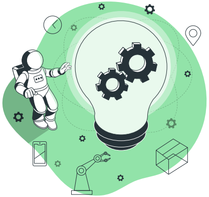
# Motivation

- Acquire accurate timestamps of keystrokes for input sequences
- Depend on bigrams, syllables, words, keyboard layout and typing experience

# Motivation

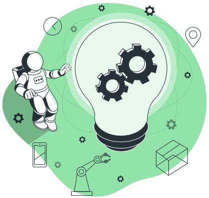
- Acquire accurate timestamps of keystrokes for input sequences
- Depend on bigrams, syllables, words, keyboard layout and typing experience
- Exploit timing characteristics to learn information about the user or the input
  - Infer typed sentences
  - Recover passphrases

# Interrupt-timing Attacks



- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps [11]

# Interrupt-timing Attacks



- Idea: Continuously acquire a high-resolution timestamp and monitor differences between subsequent timestamps [11]

# Interrupt-timing Attacks

- Look at how much time has passed since the last measurement

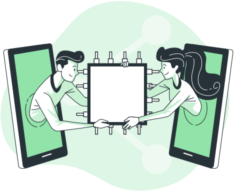
# Interrupt-timing Attacks

- Look at how much time has passed since the last measurement
- **Significant differences** occur when the process is **interrupted**

# Interrupt-timing Attacks

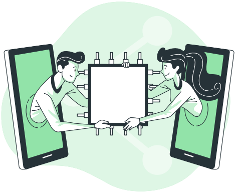
- Look at how much time has passed since the last measurement
- **Significant differences** occur when the process is **interrupted**
- More time the operating system consumes to handle the interrupt
  → higher timing difference

# Case Studies

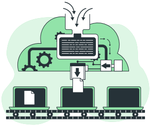


Covert Channel

# Case Studies

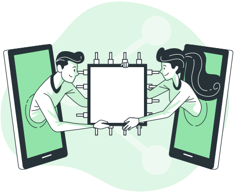


Covert Channel



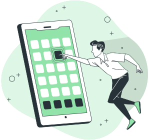
User and URL Classification

# Case Studies



Covert Channel

User and URL Classification



Touchscreen Interaction

# PIN input

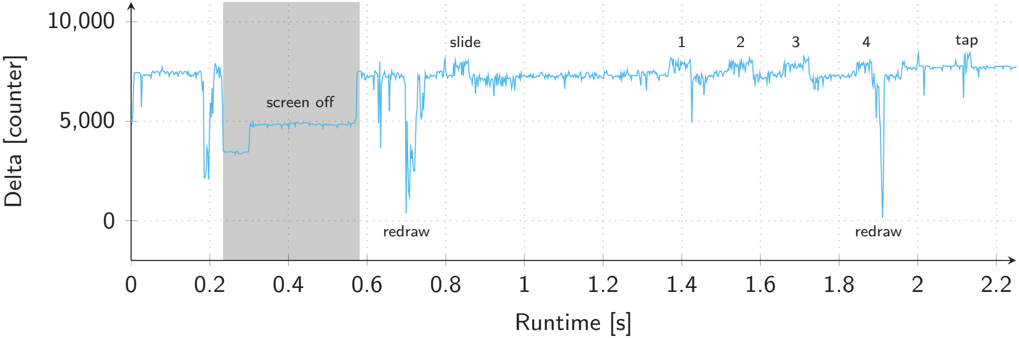
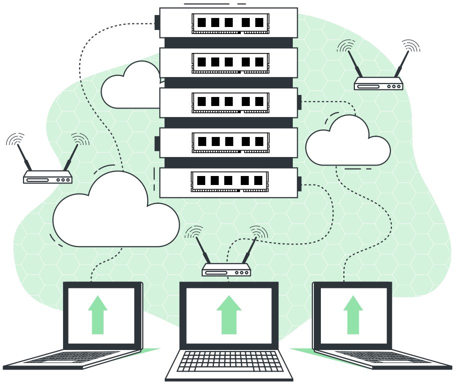


**Figure 3:** Keystroke timing attack running in the Firefox browser on the Xiaomi Redmi Note 3. While the user locked the screen, the application still detects keystrokes as long as it is executed on the last used tab.

# Publication

**Lipp, M.**, Gruss, D., Schwarz, M., Bidner, D., Maurice, C.-m.-t.-n., Mangard, S., "Practical Keystroke Timing Attacks in Sandboxed JavaScript". In: *ESORICS*. 2017



# Nethammer: Remote Rowhammer

# Remote Rowhammer Attack

- Rowhammer always required local code execution.
- Is Rowhammer possible **without** any attacker-controlled code?

# Nethammer

- Sending as many small UDP packets as possible, triggering memory accesses
- Artificial setup: bit flips every 350 ms.
- With Intel CAT, up to 25 bit flips in 15 minutes.

# Results

- Automatic classification of **memory-controller policies**
- Showed that TRR is insufficient in mitigating Rowhammer attacks

# Publication

**Lipp, M.**, Schwarz, M., Raab, L., Lamster, L., Aga, M. T., Maurice, C., Gruss, D., "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *SILM Workshop*. 2020

# My PhD in Numbers

23 Publications

(14 Tier 1, 2 Journals)

23 Publications

(14 Tier 1, 2 Journals)

7 First Author

(3 Tier 1, 1 Journal)

23 Publications

(14 Tier 1, 2 Journals)

7 First Author

(3 Tier 1, 1 Journal)

4 under submission

23 Publications

(14 Tier 1, 2 Journals)

7 First Author

(3 Tier 1, 1 Journal)

4 under submission

32 Talks

23 Publications

(14 Tier 1, 2 Journals)

7 First Author

(3 Tier 1, 1 Journal)

4 under submission

32 Talks

10 Awards + 11 CVEs

Met, worked and made friends with many incredible kind and talented people

# Conclusion

# Conclusion

- **Demonstrate** how **microarchitectural optimizations** can be **exploited** from software
- Typically **require** complex mitigations coming with a non-negligible performance impact
- Require **rethinking** on a microarchitectural level

. . . or in other words . . .

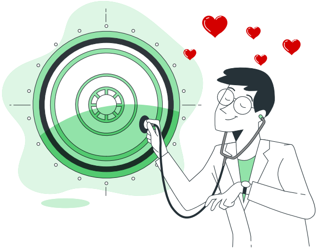
# Cooking

Cooking

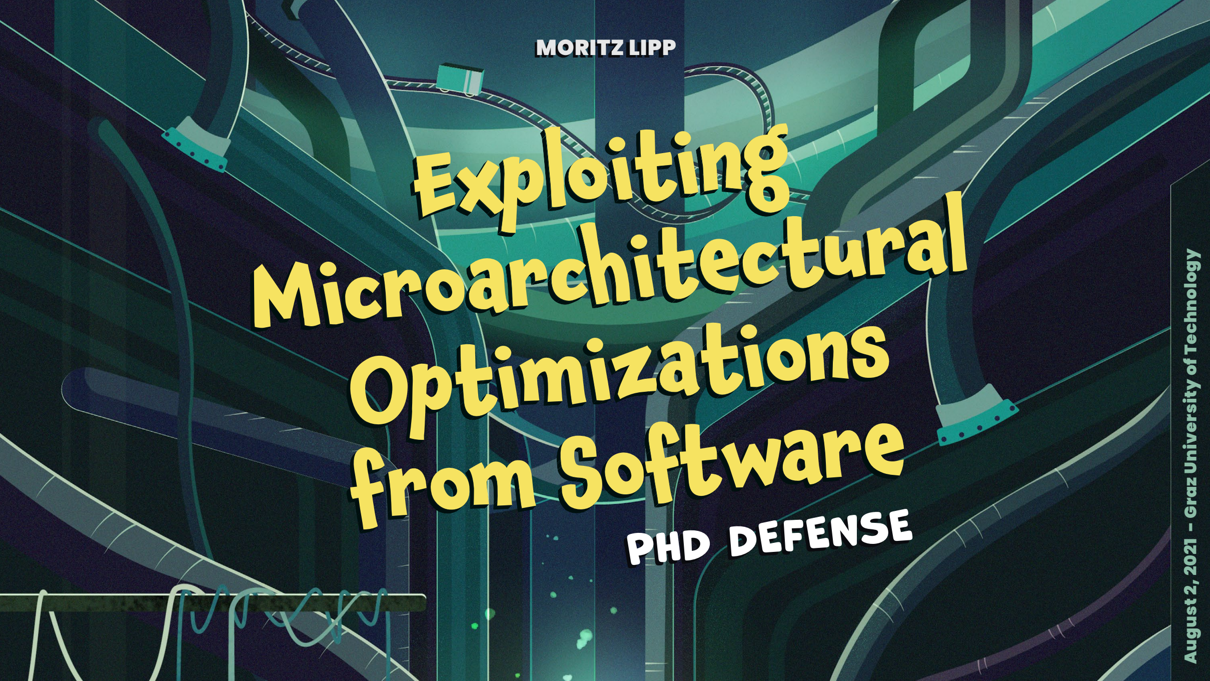
Beekeeping

Cooking

Beekeeping



Side Channels



MORITZ LIPP

# Exploiting Microarchitectural Optimizations from Software

PHD DEFENSE

August 2, 2021 – Graz University of Technology

# References i

[1] Gruss, D., **Lipp, M.**, Schwarz, M., Fellner, R., Maurice, C., Mangard, S., "KASLR is Dead: Long Live KASLR". In: *ESSoS*. 2017.

[2] Gruss, D., Maurice, C., Fogh, A., **Lipp, M.**, Mangard, S., "Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR". In: *ACM CCS*. 2016.

[3] Hund, R., Willems, C., Holz, T., "Practical Timing Side Channel Attacks against Kernel Space ASLR". In: *IEEE S&P*. 2013.

[4] Inoue, K., Ishihara, T., Murakami, K., "Way-predicting set-associative cache for high performance and low energy consumption". In: *Symposium on Low Power Electronics and Design*. 1999.

# References ii

[5] Jang, Y., Lee, S., Kim, T., “Breaking Kernel Address Space Layout Randomization with Intel TSX”. In: *ACM CCS*. 2016.

[6] **Lipp, M.**, Gruss, D., Schwarz, M., Bidner, D., Maurice, C.-m.-t.-n., Mangard, S., “Practical Keystroke Timing Attacks in Sandboxed JavaScript”. In: *ESORICS*. 2017.

[7] **Lipp, M.**, Hadžić, V., Schwarz, M., Perais, A., Maurice, C., Gruss, D., “Take a Way: Exploring the Security Implications of AMD’s Cache Way Predictors”. In: *AsiaCCS*. 2020.

[8] **Lipp, M.**, Kogler, A., Oswald, D., Schwarz, M., Easdon, C., Canella, C., Gruss, D., “PLATYPUS: Software-based Power Side-Channel Attacks on x86”. In: *IEEE S&P*. 2021.

# References iii

[9] **Lipp, M.**, Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M., "Meltdown: Reading Kernel Memory from User Space". In: *USENIX Security Symposium*. 2018.

[10] **Lipp, M.**, Schwarz, M., Raab, L., Lamster, L., Aga, M. T., Maurice, C., Gruss, D., "Nethammer: Inducing Rowhammer Faults through Network Requests". In: *SILM Workshop*. 2020.

[11] Schwarz, M., **Lipp, M.**, Gruss, D., Weiser, S., Maurice, C.-m.-t.-n., Spreitzer, R., Mangard, S., "KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks". In: *NDSS*. 2018.

# References iv

[12] Tatar, A., Krishnan, R., Athanasopoulos, E., Giuffrida, C., Bos, H., Razavi, K., "Throwhammer: Rowhammer Attacks over the Network and Defenses". In: *USENIX ATC*. 2018.