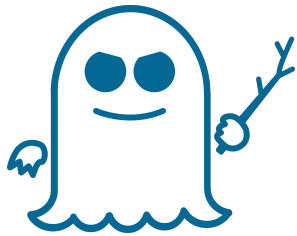
CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

CPU Fuzzing

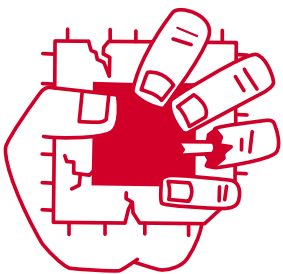
Automatic Discovery of Microarchitectural Attacks

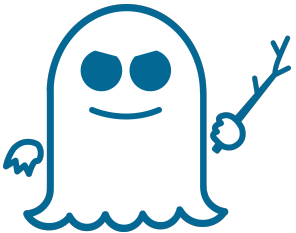
Daniel Weber, Michael Schwarz | May 12, 2023


FIREEYE CYBER
THREAT MAP
DEVELOPING STORY
COMPUTER CHIP FLAWS IMPACT BILLIONS OF DEVICES
LIVE
CNN
DAX ▲164.69
NEWS STREAM



Daniel Weber (@weber_daniel), Michael Schwarz (@misc0110)





Daniel Weber (@weber_daniel), Michael Schwarz (@misc0110)

Microarchitectural

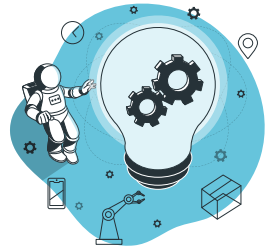
attacks

Microarchitectural attacks

How do they **work**?

Microarchitectural attacks

How do they **work**?



How can we find them **efficiently**?

Who am I?

Daniel Weber

PhD Student @ CISPA Helmholtz Center for Information Security

Research on CPU Security and Side Channels

@weber_daniel

daniel.weber@cispa.de

Who am I?

Michael Schwarz

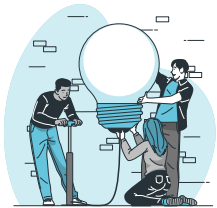
Faculty @ CISPA Helmholtz Center for Information Security

Research on CPU Security and Side Channels

@misc0110

michael.schwarz@cispa.de

Team



Osiris Research Team

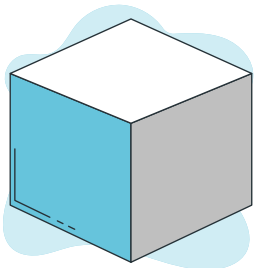
Daniel Weber, *Ahmad Ibrahim, Hamed Nemati,* ***Michael Schwarz***, *Christian Rossow*

Transynther Research Team

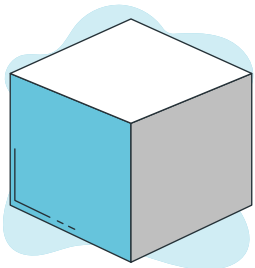
Daniel Moghimi, Moritz Lipp, Berk Sunar, ***Michael Schwarz***

MSRevelio Research Team

Andreas Kogler, ***Daniel Weber***, *Martin Haubenwallner, Moritz Lipp, Daniel Gruss,* ***Michael Schwarz***

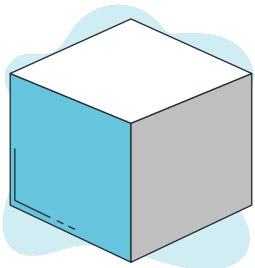


CPU



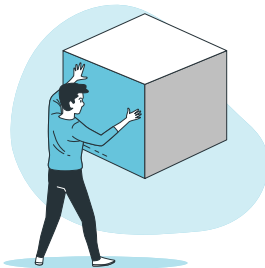
CPU

Specification (ISA)



CPU

Specification (ISA)



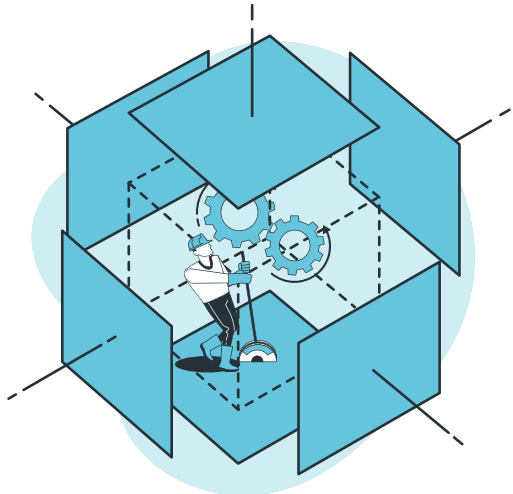
Interaction

Everything **works**
as expected

Everything **works**
as expected

No bugs

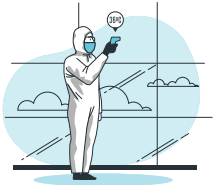
How can you attack this?



Information **leaks** through **side effects**

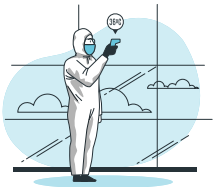
Power Consumption

Power Consumption

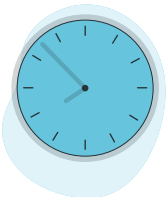


Temperature

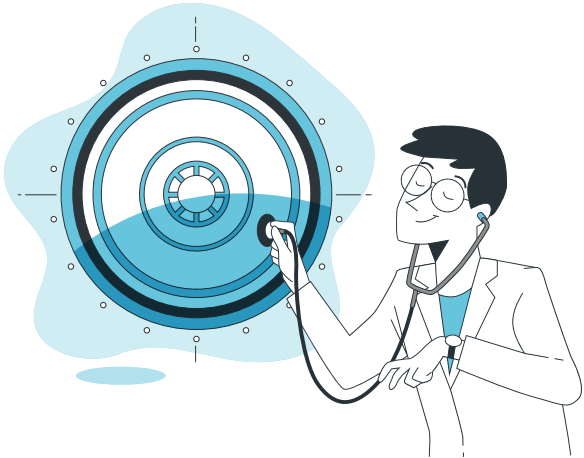
Power Consumption



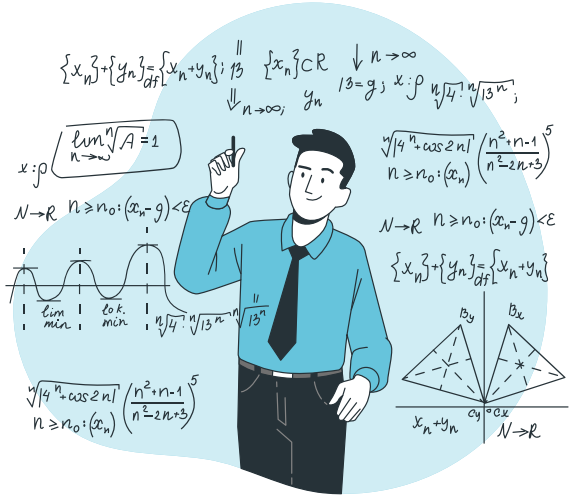
Temperature



Execution Time



Observe Device

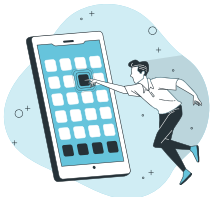


Evaluate data

Get the secrets. **Easy as that!**

Reality **is not** as easy

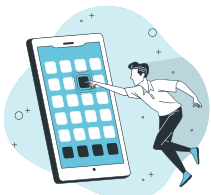
Properties



Software-only

No Physical Access required

Properties



Software-only

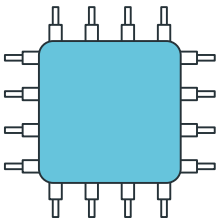
No Physical Access required

Understand Inner Workings

of the CPU

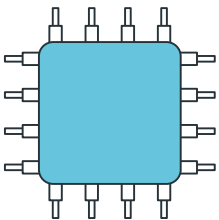
What can we observe? How does a CPU work internally?

Architecture vs Microarchitecture



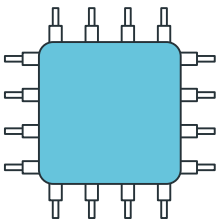
- Instruction Set Architecture (ISA) is an **abstract model** of a computer (x86, ARMv8, SPARC, …)

Architecture vs Microarchitecture



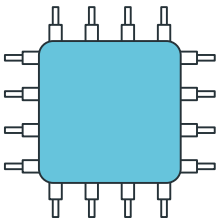
- Instruction Set Architecture (ISA) is an **abstract model** of a computer (x86, ARMv8, SPARC, …)
- **Interface** between hardware and software

Architecture vs Microarchitecture



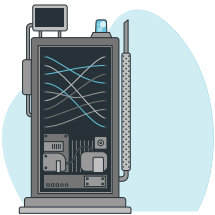
- Instruction Set Architecture (ISA) is an **abstract model** of a computer (x86, ARMv8, SPARC, …)
- **Interface** between hardware and software
- Microarchitecture is an ISA **implementation**

Architecture vs Microarchitecture



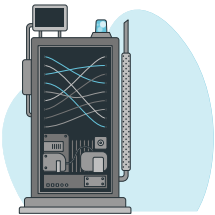
- Instruction Set Architecture (ISA) is an **abstract model** of a computer (x86, ARMv8, SPARC, …)
- **Interface** between hardware and software
- Microarchitecture is an ISA **implementation**

Optimizations



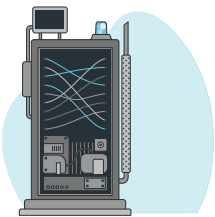
- Modern CPUs contain multiple **microarchitectural elements**

Optimizations



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

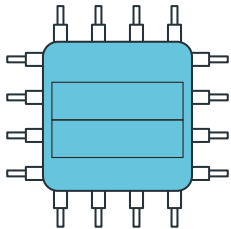
Optimizations

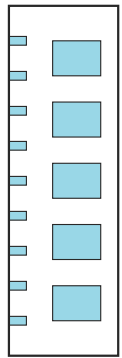


- Modern CPUs contain multiple **microarchitectural elements**
- **Transparent** for the programmer
- **Optimize** for performance, power consumption, …

CPU Optimization: Cache

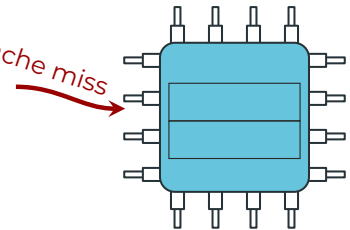
```
printf("%d", i);

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

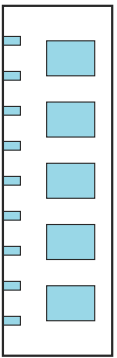




CPU Optimization: Cache

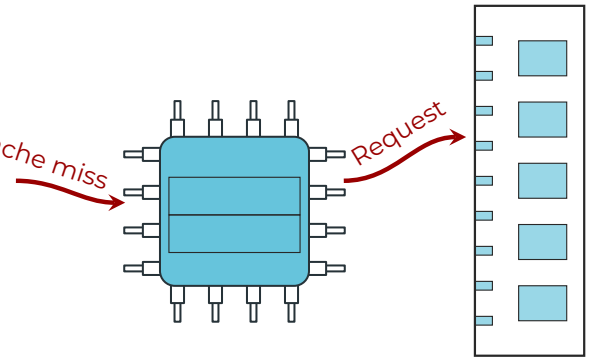
```
printf("%d", i);

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





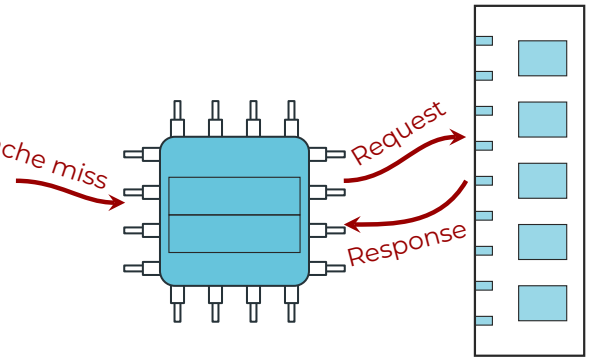
CPU Optimization: Cache

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

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



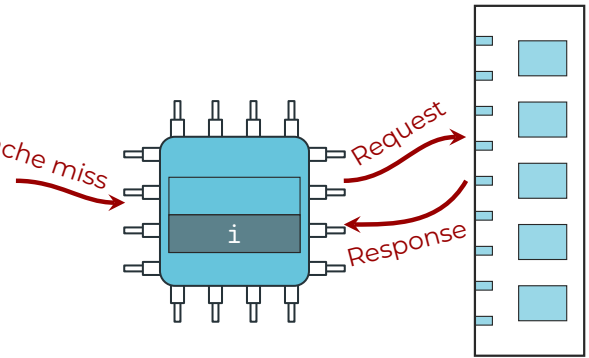
CPU Optimization: Cache

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

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

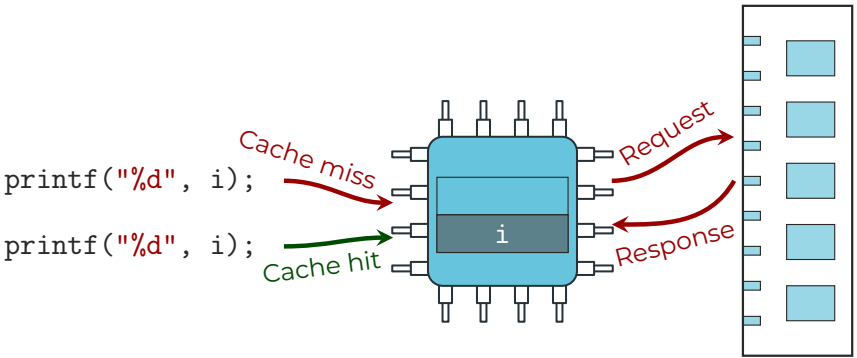


CPU Optimization: Cache

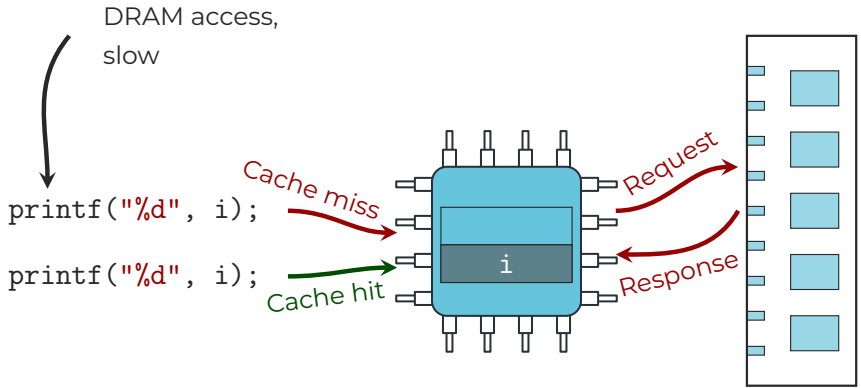
```
printf("%d", i);

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



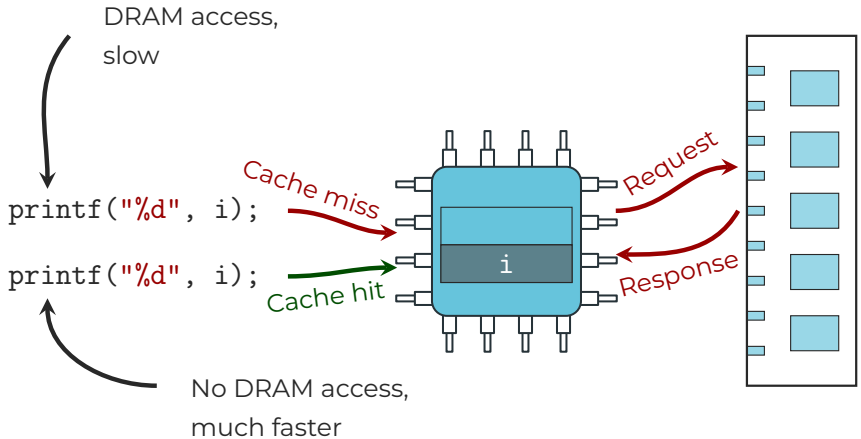
CPU Optimization: Cache



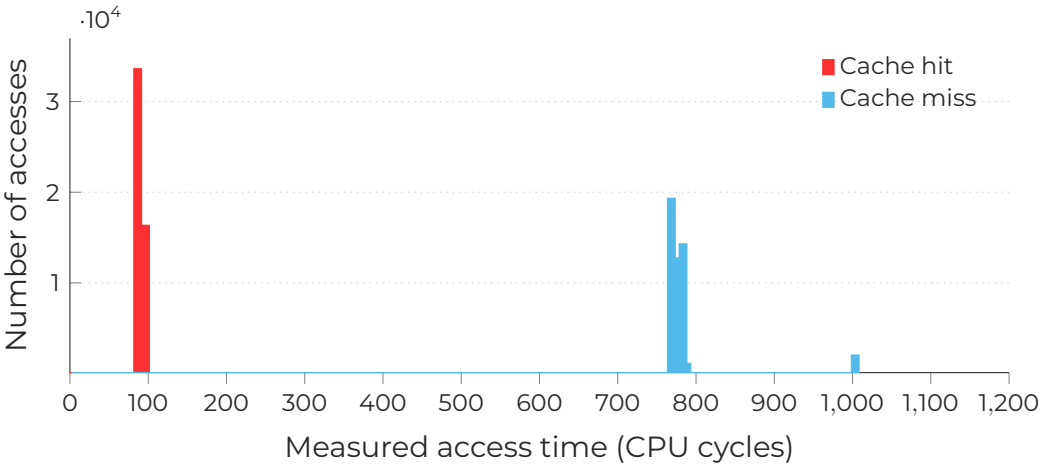
CPU Optimization: Cache



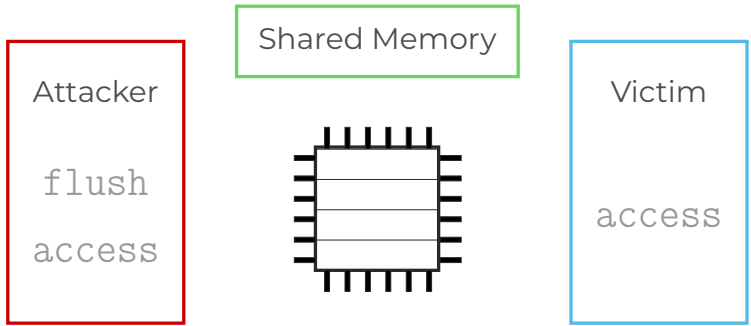
CPU Optimization: Cache



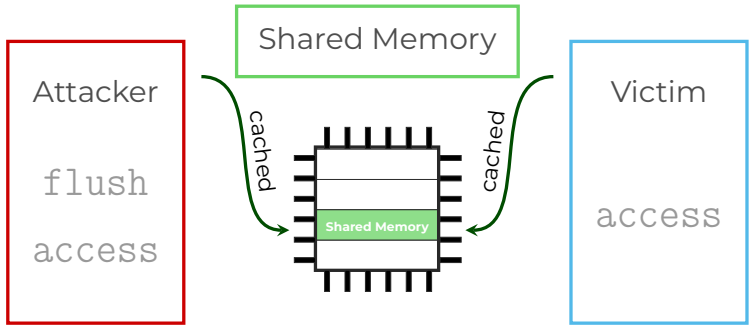
Caching speeds up Memory Accesses



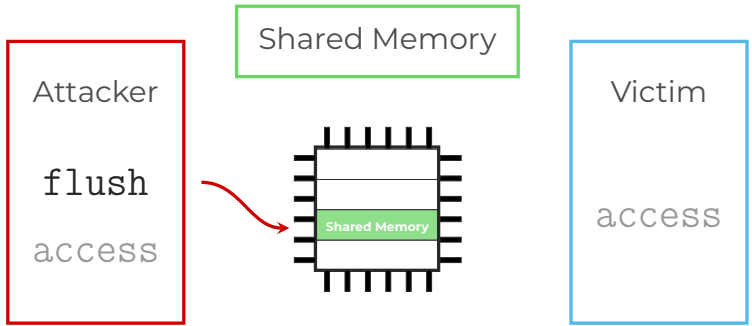
Flush+Reload



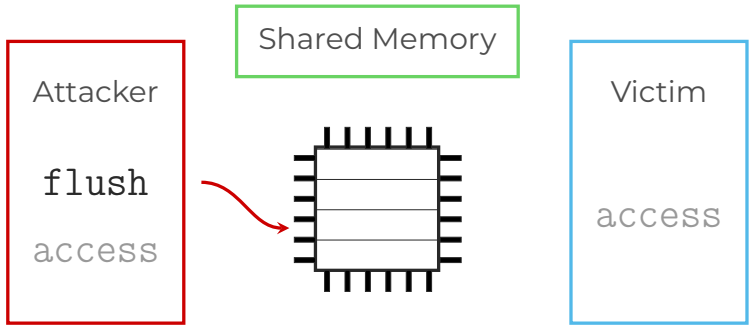
Flush+Reload



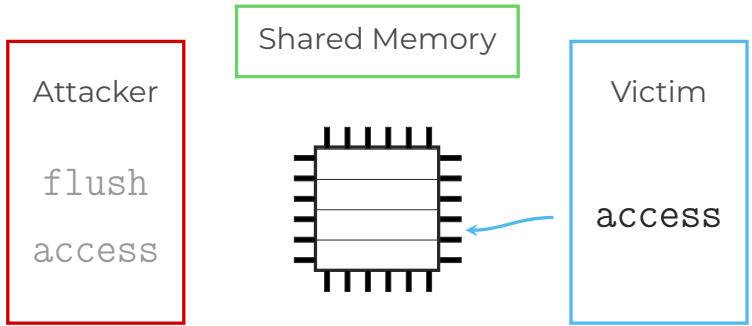
Flush+Reload



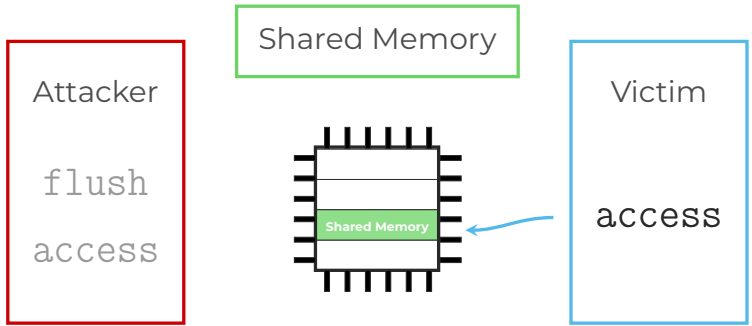
Flush+Reload



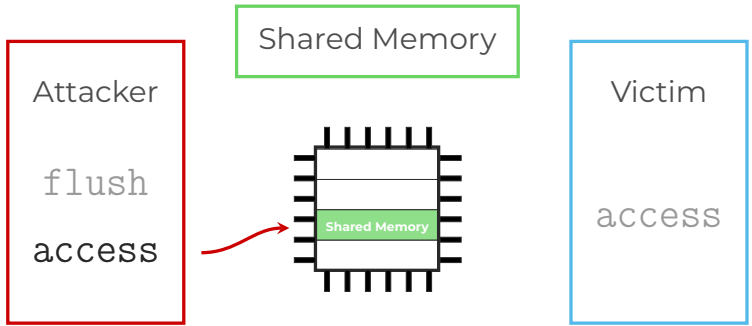
Flush+Reload



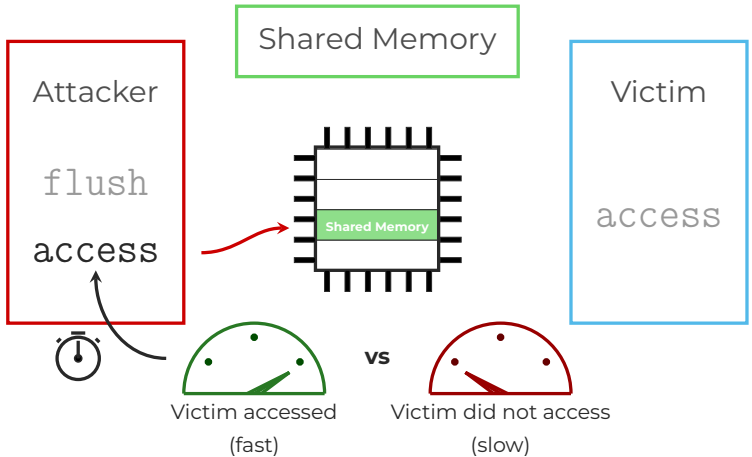
Flush+Reload



Flush+Reload



Flush+Reload



Cache Attacks

- Leak **cryptographic keys**

Cache Attacks

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

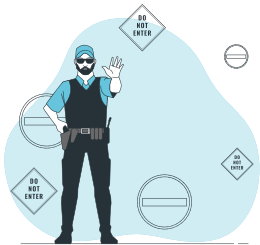
Cache Attacks

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

Cache Attacks

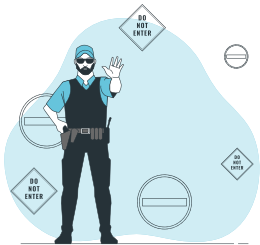
- Leak **cryptographic keys**
- Leak information on **co-located virtual machines**
- **Monitor** function calls of other applications
- Build **covert communication channels**
- …

Motivation

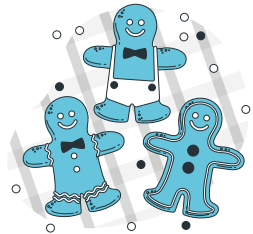


Leakage of **1-2** Instructions

Motivation



Leakage of **1-2** Instructions

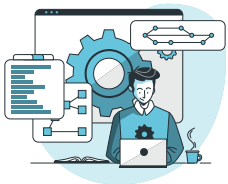


There are **>3000** instructions

How do you find **these** attacks?

Read documentation and patents

Read documentation and patents



Develop ideas

Read documentation and patents

Develop ideas

Test ideas and start over

New CPU, **New** Features

New CPU, **New** Features

Repeat from the beginning

Finding side channels and vulnerabilities is a **complex** and **time-consuming** process

Can we do **better**?

Automation for Side-Channel Search

- How can you find bugs in **software** in an automated way?

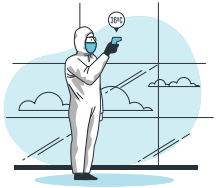
Automation for Side-Channel Search

- How can you find bugs in **software** in an automated way?
- **Fuzzing** is an automated software testing method
  - Inject invalid, malformed, unexpected input to a program

Automation for Side-Channel Search

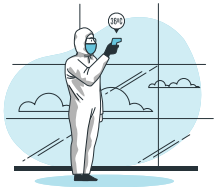
- How can you find bugs in **software** in an automated way?
- **Fuzzing** is an automated software testing method
  - Inject invalid, malformed, unexpected input to a program
- Can we **apply a similar approach** do find microarchitectural side-channels?

Challenges when Fuzzing CPUs



Limited Feedback

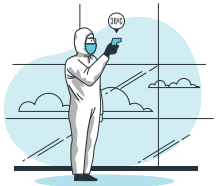
Challenges when Fuzzing CPUs



Limited Feedback

Dirty States

Challenges when Fuzzing CPUs



Limited Feedback

Dirty States

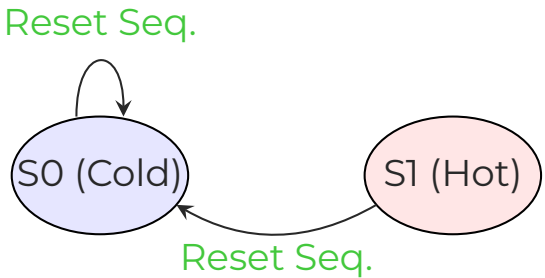
System Noise

Side Channel Search

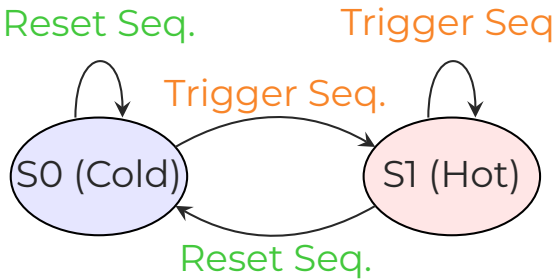
Can we **automatically decide** whether we **found a side channel?**

Modelling Side Channels

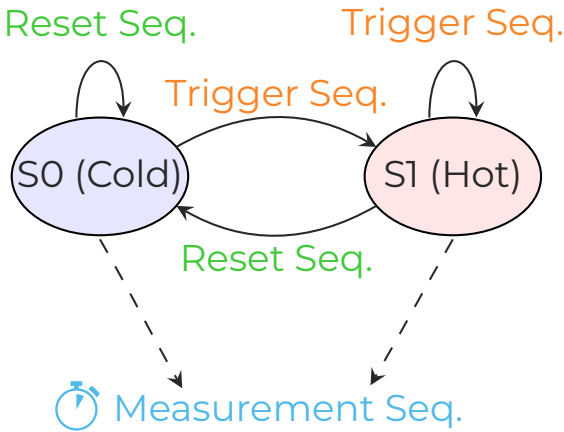
Modelling Side Channels



Modelling Side Channels



Modelling Side Channels



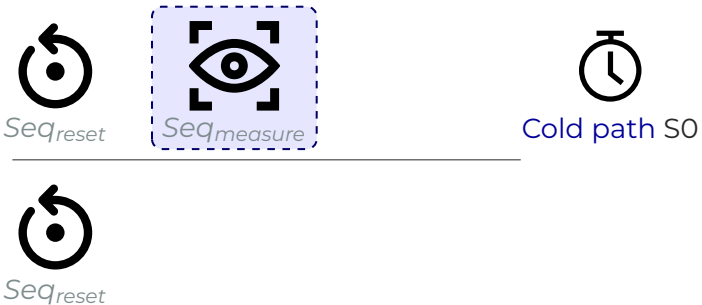
Testing Potential Side Channels

Testing Potential Side Channels

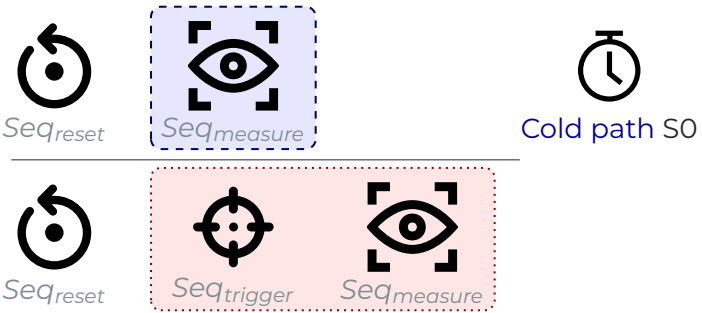
Testing Potential Side Channels

Cold path S0

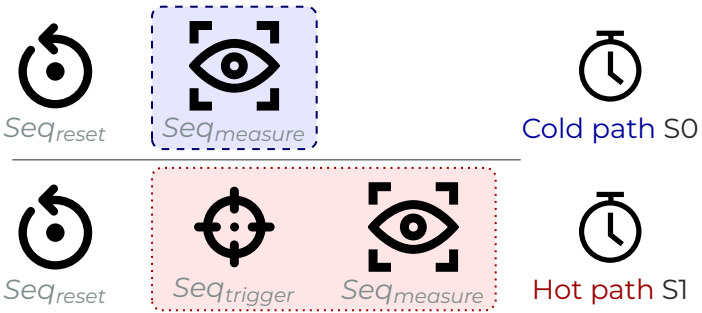
Testing Potential Side Channels



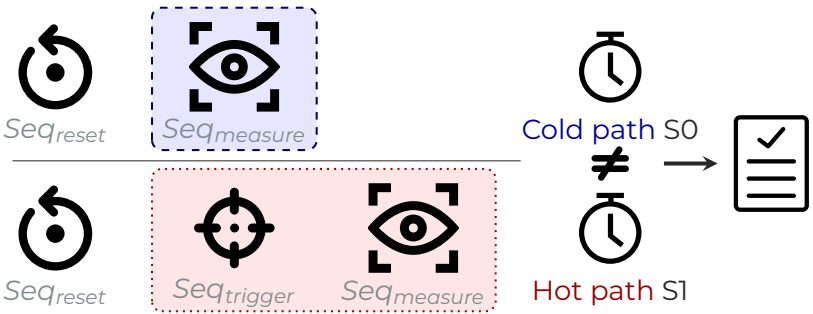
Testing Potential Side Channels



Testing Potential Side Channels

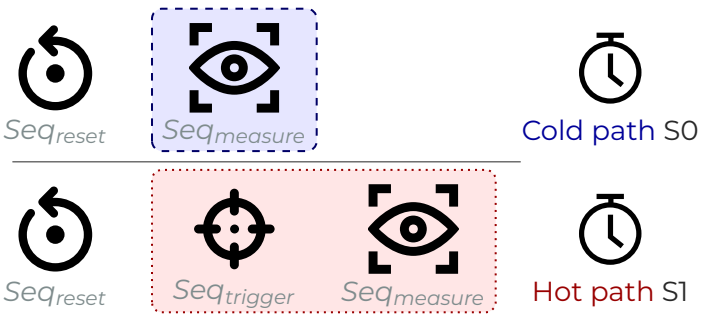


Testing Potential Side Channels



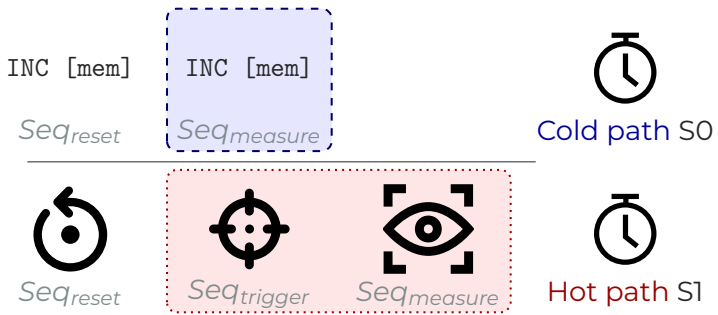
Testing Potential Side Channels

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



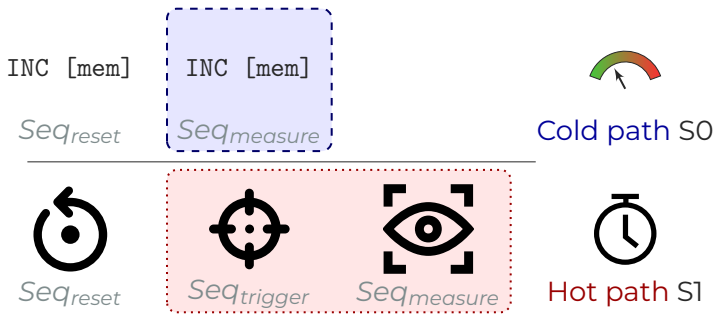
Testing Potential Side Channels

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



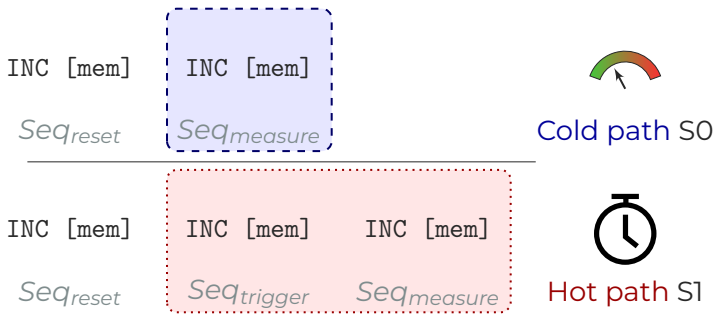
Testing Potential Side Channels

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



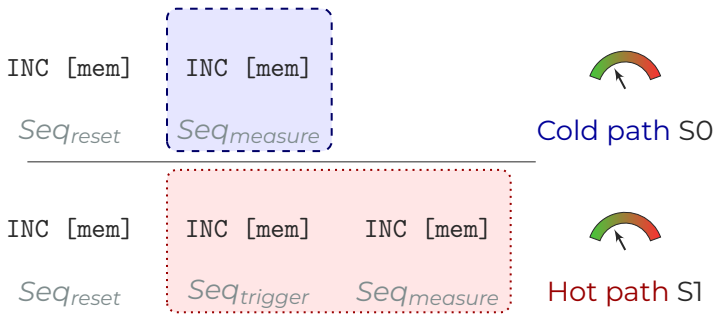
Testing Potential Side Channels

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



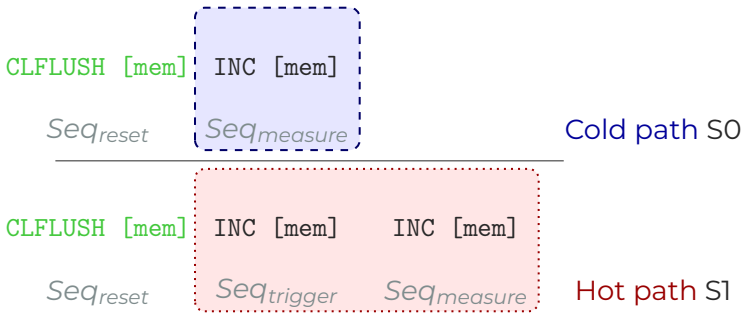
Testing Potential Side Channels

Example 1: $Seq_{measure} = Seq_{trigger} = Seq_{reset} =$ `INC [mem]`



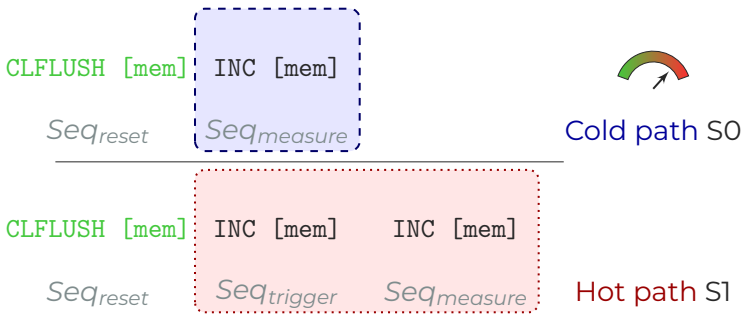
Testing Potential Side Channels

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem];`
$Seq_{reset} =$ `CLFLUSH [mem]`



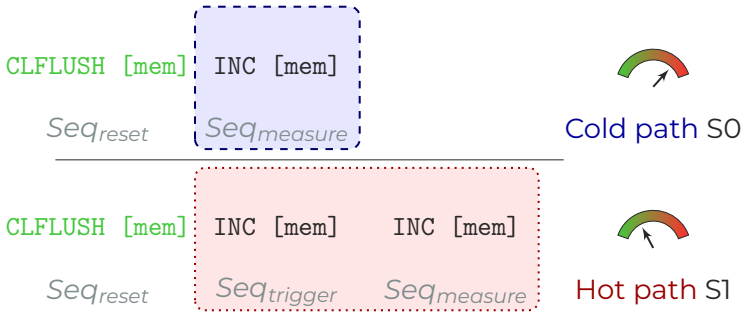
Testing Potential Side Channels

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem];`
$Seq_{reset} =$ `CLFLUSH [mem]`



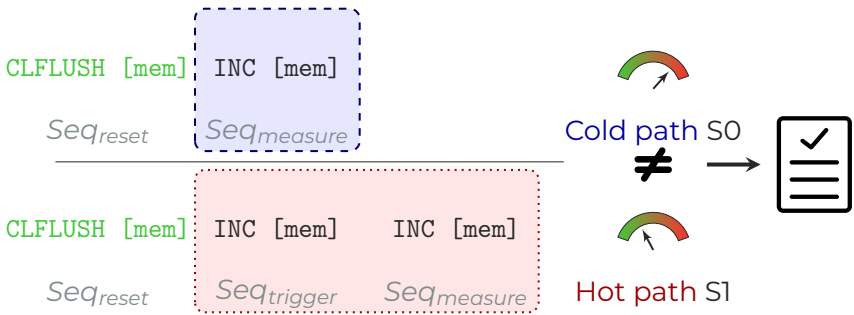
Testing Potential Side Channels

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem];`
$Seq_{reset} =$ `CLFLUSH [mem]`



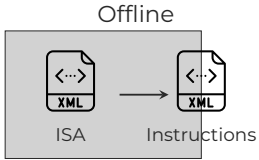
Testing Potential Side Channels

Example 2: $Seq_{measure} = Seq_{trigger} =$ `INC [mem];`
$Seq_{reset} =$ `CLFLUSH [mem]`

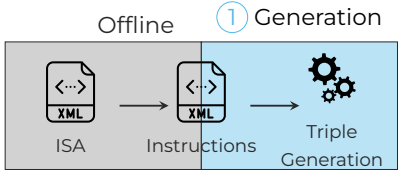


Let’s Fuzz for Side Channels

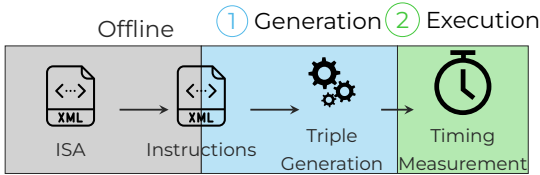
Osiris – Fuzzing x86 CPUs for Side Channels



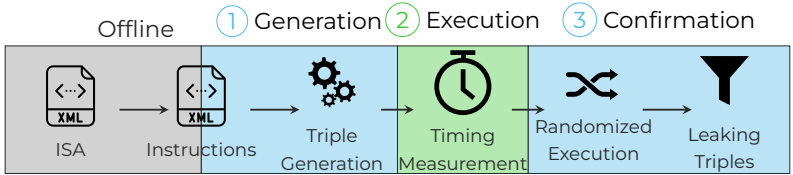
Osiris – Fuzzing x86 CPUs for Side Channels



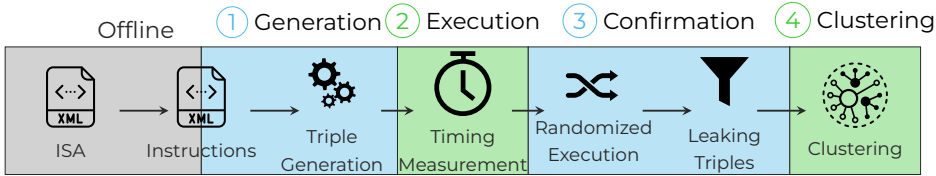
Osiris – Fuzzing x86 CPUs for Side Channels



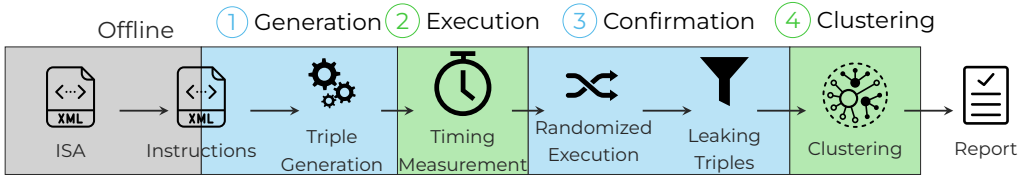
Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Results Overview

4 days

Osiris – Results Overview

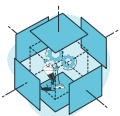
4 days

2 side channels reproduced

Osiris – Results Overview

4 days

2 side channels reproduced

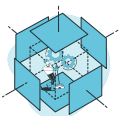


4 new side channels

Osiris – Results Overview

4 days

2 side channels reproduced



4 new side channels

1 new attack

Kernel ASLR (KASLR)

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

Kernel ASLR (KASLR)

- Many exploits rely on the **knowledge of the memory location** of a certain function
- OS Kernels are **protected by KASLR**

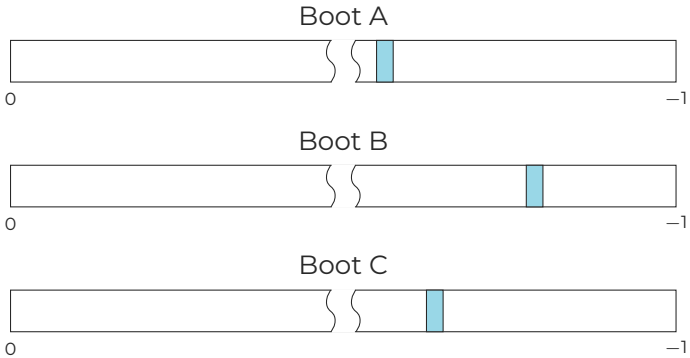
Kernel ASLR (KASLR)

- Many exploits rely on the **knowledge of the memory location** of a certain function
- OS Kernels are **protected by KASLR**
- Memory location of the kernel is **randomized**

Kernel ASLR (KASLR)

- Many exploits rely on the **knowledge of the memory location** of a certain function
- OS Kernels are **protected by KASLR**
- Memory location of the kernel is **randomized**
- Attacker **do not know where** to attack

KASLR: Kernel Address Space Layout Randomization



- Kernel is loaded to a different offset on every boot

Findings – FlushConflict

- **KASLR break** based on MOVNT and transient execution

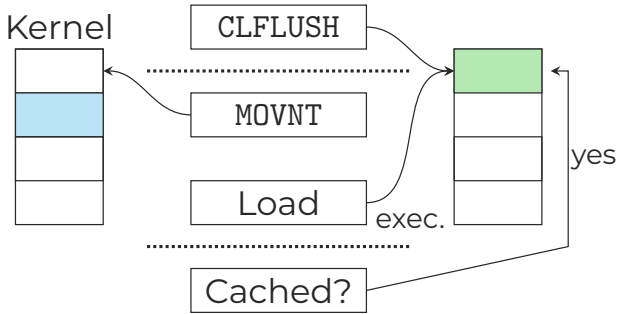
Findings – FlushConflict

- **KASLR break** based on MOVNT and transient execution
- Works **on new Intel CPUs** (tested up to Alder Lake)

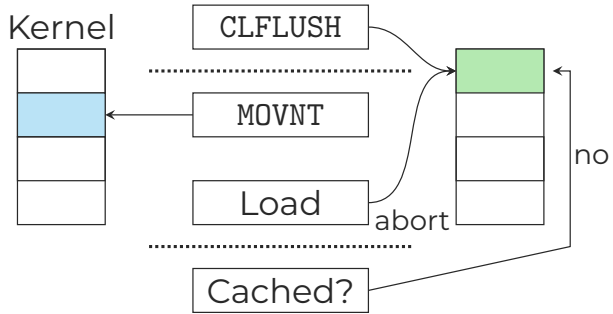
Findings – FlushConflict

- **KASLR break** based on MOVNT and transient execution
- Works **on new Intel CPUs** (tested up to Alder Lake)
- Breaks KASLR reliably in **136ms (average)**

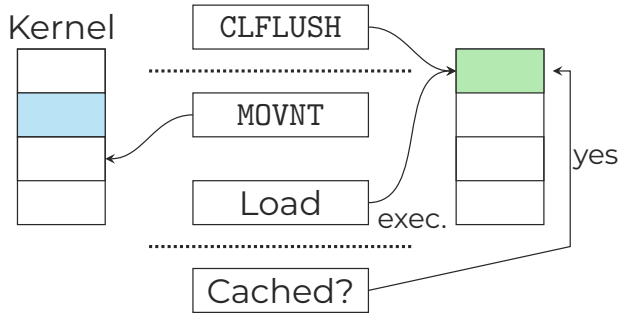
Findings – FlushConflict



Findings – FlushConflict



Findings – FlushConflict



FlushConflict Live Demo

Osiris – Lessons Learned

- We can **automatically search** for side channels

Osiris – Lessons Learned

- We can **automatically search** for side channels
- **Scaling** such tools is challenging

Osiris – Lessons Learned

- We can **automatically search** for side channels
- **Scaling** such tools is challenging

→ Osiris prototype is currently limited to 1 instruction per sequence

Can we find **more powerful attacks?**

Can we find **more powerful attacks?**

Can we **leak actual data** instead of meta data?

CPU Optimization: Lazy Error Handling

- With **Meltdown**, we exploit out-of-order execution to leak the content of **inaccessible** kernel addresses

CPU Optimization: Lazy Error Handling

- With **Meltdown**, we exploit out-of-order execution to leak the content of **inaccessible** kernel addresses
  - → Faulty state is never architecturally visible

CPU Optimization: Lazy Error Handling

- With **Meltdown**, we exploit out-of-order execution to leak the content of **inaccessible** kernel addresses
  - → Faulty state is never architecturally visible
  - → Use a side-channel (Flush+Reload) to make it visible

CPU Optimization: Lazy Error Handling

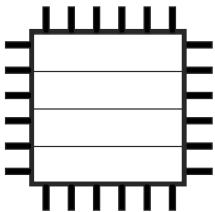
- With **Meltdown**, we exploit out-of-order execution to leak the content of **inaccessible** kernel addresses
  - → Faulty state is never architecturally visible
  - → Use a side-channel (Flush+Reload) to make it visible
- Many different attack variants:
  - → Fallout, ZombieLoad, CrossTalk, …

Meltdown-type Attacks

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = faulting[0]
```



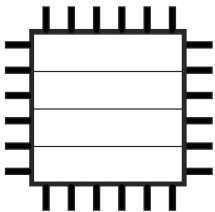
Meltdown-type Attacks

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

```
char value = faulting[0]
```

Fault



Meltdown-type Attacks

User Memory

	A	B
C	D	E
F	G	H
I	J	K
L	M	N
O	P	Q
R	S	T
U	V	W
X	Y	Z

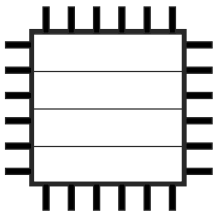
```
char value = faulting[0]
```

Fault

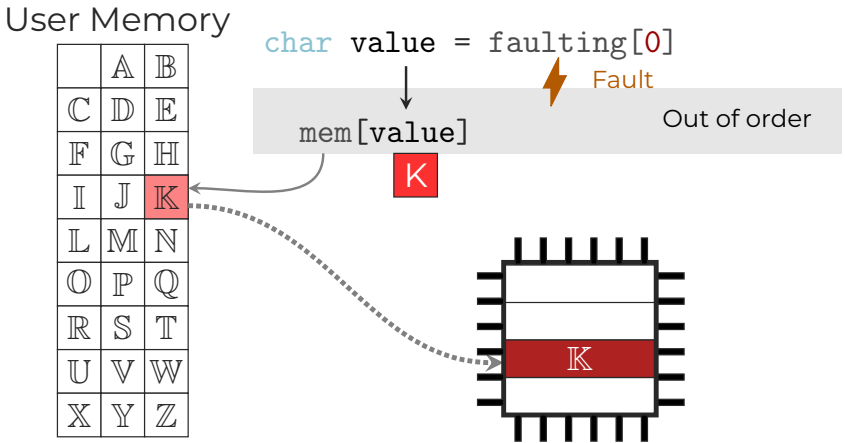
Out of order

```
mem[value]
```

K



Meltdown-type Attacks

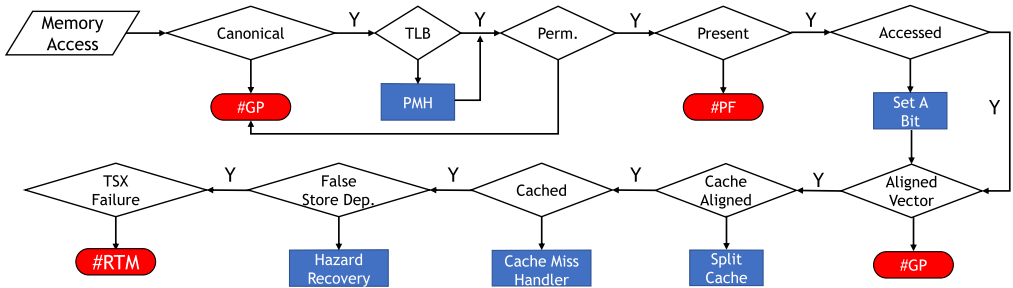


Memory Access Checks (simplified)

- Many possibilities for faults and microcode assists

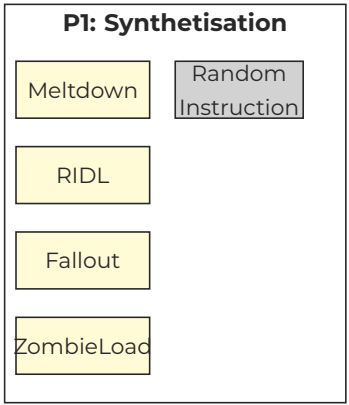
Memory Access Checks (simplified)

- Many possibilities for faults and microcode assists

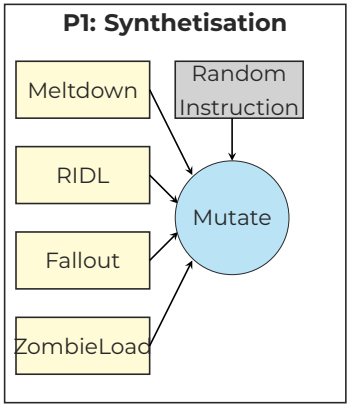


Fuzzing for Meltdown-type Attacks

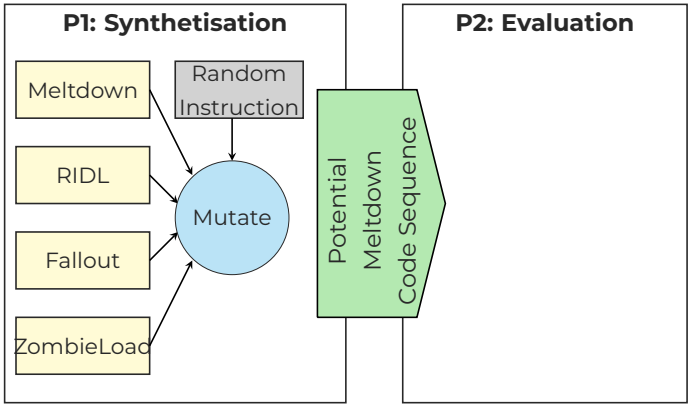
Transynther – Fuzzing for Meltdown-type Attacks



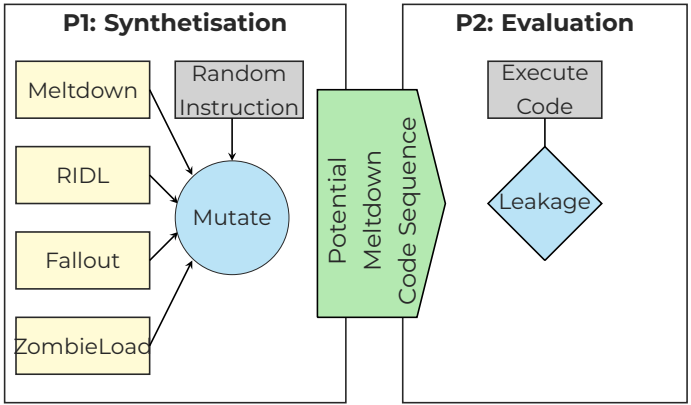
Transynther – Fuzzing for Meltdown-type Attacks



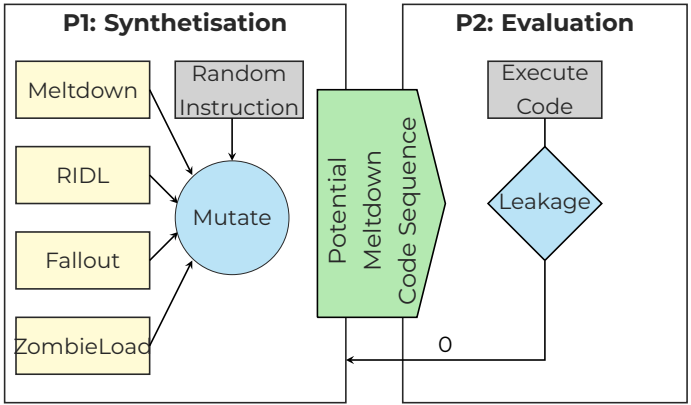
Transynther – Fuzzing for Meltdown-type Attacks



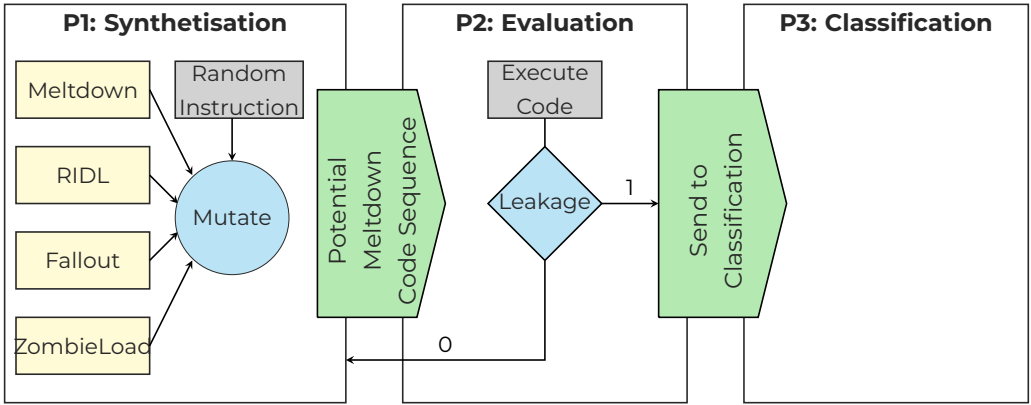
Transynther – Fuzzing for Meltdown-type Attacks



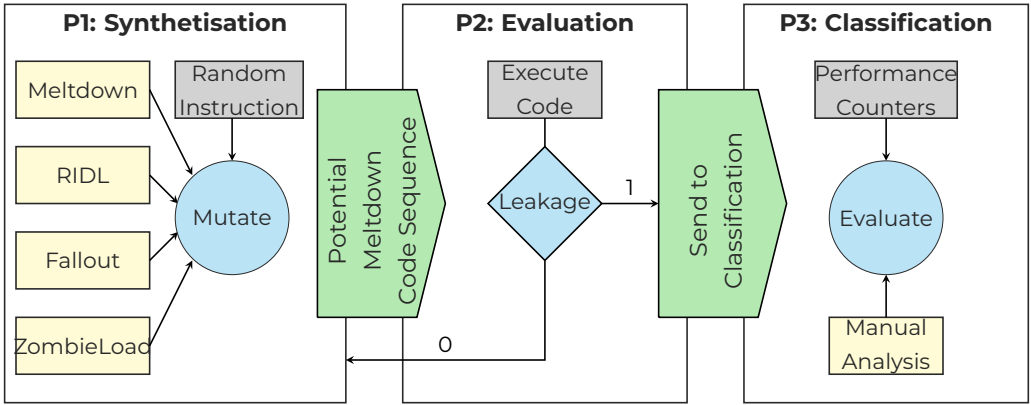
Transynther – Fuzzing for Meltdown-type Attacks



Transynther – Fuzzing for Meltdown-type Attacks



Transynther – Fuzzing for Meltdown-type Attacks



Transynther – Results Overview

26 hours

Transynther – Results Overview

26 hours

100 unique leakage patterns

Transynther – Results Overview

26 hours

100 unique leakage patterns

7 attacks reproduced

Transynther – Results Overview

26 hours

100 unique leakage patterns

7 attacks reproduced

1 new vulnerability

Transynther – Results Overview

26 hours

100 unique leakage patterns

7 attacks reproduced

1 new vulnerability

1 regression

Findings – Reenabling MDS Attack

- Medusa: new variant of ZombieLoad

Findings – Reenabling MDS Attack

- Medusa: new variant of ZombieLoad
- Leaks from **write-combining buffer**, i.e., `REP MOV`

Findings – Reenabling MDS Attack

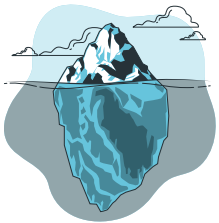
- Medusa: new variant of ZombieLoad
- Leaks from **write-combining buffer**, i.e., `REP MOV`
- Used for fast memory copy, e.g., in OpenSSL or kernel

Findings – Reenabling MDS Attack

- Medusa: new variant of ZombieLoad
- Leaks from **write-combining buffer**, i.e., `REP MOV`
- Used for fast memory copy, e.g., in OpenSSL or kernel

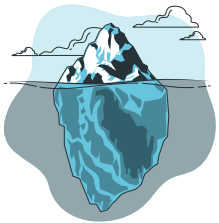
→ Leaked RSA key while decoding in OpenSSL

Findings – MDS Attack on Ice Lake



- Intel's Ice Lake CPUs were labeled **MDS-resistant**

Findings – MDS Attack on Ice Lake



- Intel's Ice Lake CPUs were labeled **MDS-resistant**

→ Transynther found a working MDS attack

  - Variant of a Fallout attack

Transynther – Lessons Learned

- We can **automatically search** for transient-execution attacks

Transynther – Lessons Learned

- We can **automatically search** for transient-execution attacks
- **Post-analysis** can also be automated

Transynther – Lessons Learned

- We can **automatically search** for transient-execution attacks
- **Post-analysis** can also be automated
- **Regression bugs can be found** using automated tools

Are there also **architectural bugs**?

Are there also **architectural bugs**?

What about **unknown instructions**?

Finding Unknown Instructions

- **Sandsifter**[a] searches for undocumented x86 instructions

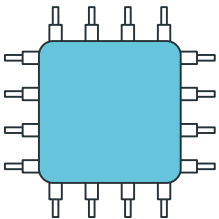
Finding Unknown Instructions

- **Sandsifter**[a] searches for undocumented x86 instructions
  - Trick to check for valid instruction bytes using page faults
- **Haruspex**[b] uses speculative execution and performance counters

[a] Sandsifter (Christopher Domas): `https://github.com/xoreaxeaxeax/sandsifter`
[b] Haruspex (Can Bölük): `https://github.com/can1357/haruspex`

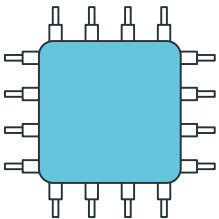
What else is **left**?

Model-Specific Registers and their Hidden Secrets



- Model-Specific Registers (MSRs) are **special CPU control registers**

Model-Specific Registers and their Hidden Secrets

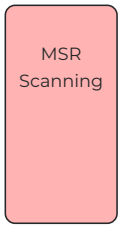


- Model-Specific Registers (MSRs) are **special CPU control registers**
- VIA C3 CPU had backdoor access[a] hidden behind an **undocumented MSR bit**

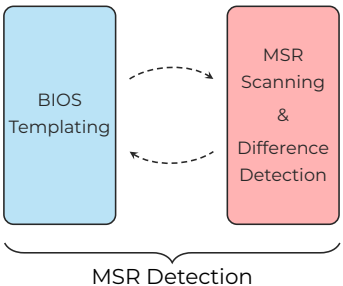
[a] GOD MODE unlocked: Hardware backdoors in x86 CPUs (Christopher Domas – BlackHat USA '18)

Searching for Undocumented MSRs

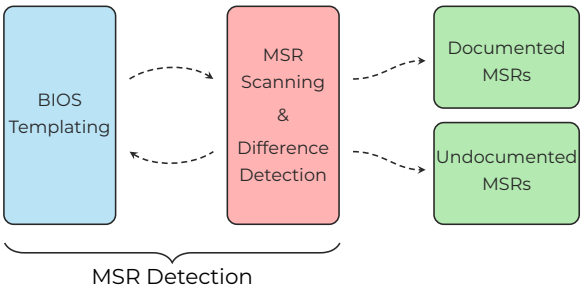
MSRevelio – Overview



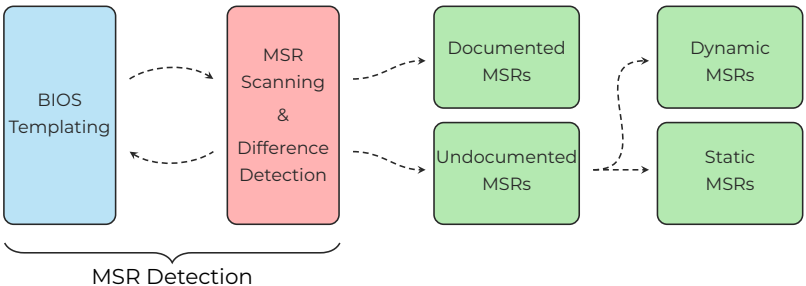
MSRevelio – Overview



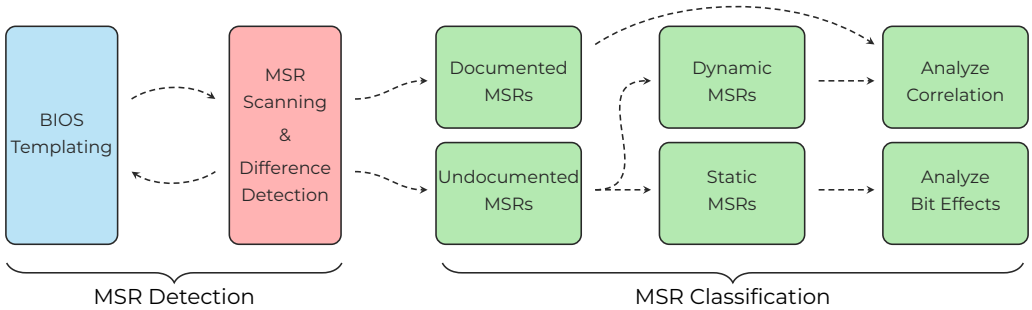
MSRevelio – Overview



MSRevelio – Overview



MSRevelio – Overview



MSRevelio – Results Overview

5890 undocumented MSRs

MSRevelio – Results Overview

5890 undocumented MSRs

3 MSRs as attack **mitigation**

MSRevelio – Results Overview

5890 undocumented MSRs

3 MSRs as attack **mitigation**

1 MSR allows

TOCTOU vulnerability

MSRevelio – Results Overview

5890 undocumented MSRs

3 MSRs as attack **mitigation**

1 MSR allows
TOCTOU vulnerability

New MSRs hint towards vulnerabilities

Findings – Undisclosed Attacks

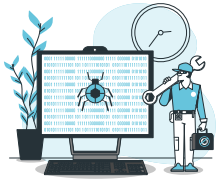
- MSRs often used to introduce **vulnerability fixes**

Findings – Undisclosed Attacks

- MSRs often used to introduce **vulnerability fixes**
- MSRs exist **before public disclosure**

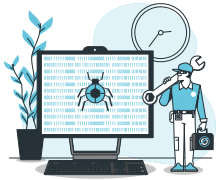
→ Useful for 1-Day Exploits

Findings – Attack Mitigation



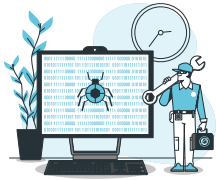
- Mitigate **prefetch side-channel attacks**

Findings – Attack Mitigation



- Mitigate **prefetch side-channel attacks**
- Reduce **Crosstalk leakage**

Findings – Attack Mitigation



- Mitigate **prefetch side-channel attacks**
- Reduce **Crosstalk leakage**
- Reduce **Medusa leakage**

MSRevelio – Lessons Learned

- Searching for **unknown behavior is hard**

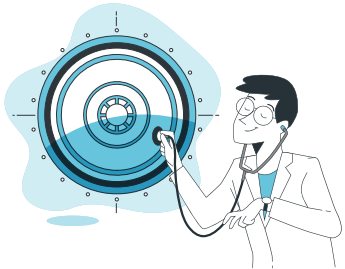
MSRevelio – Lessons Learned

- Searching for **unknown behavior is hard**
- We can **automate the search** for undocumented MSR behavior

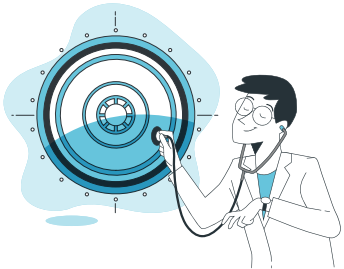
MSRevelio – Lessons Learned

- Searching for **unknown behavior is hard**
- We can **automate the search** for undocumented MSR behavior
- Automation allows **tracing changes between releases**
  (cf. Transynther)

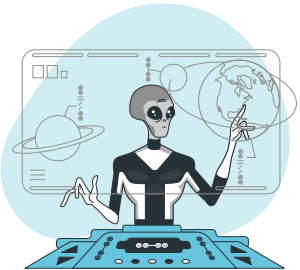
Conclusion



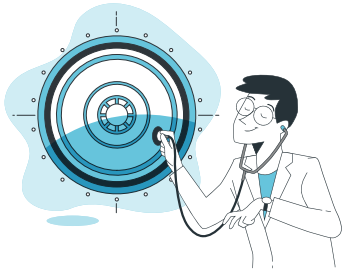
Automated
side-channel discovery



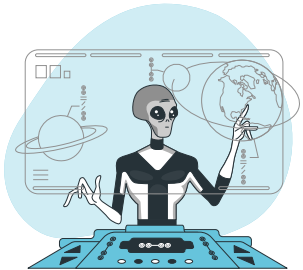
Automated
side-channel discovery



Automated
transient-execution attack discovery



Automated
side-channel discovery



Automated
transient-execution attack discovery

Automated detection of
undocumented MSRs

CPU Fuzzing – Lessons Learned

- **Scalability**: Automation is easier than manual efforts (even if limited)

CPU Fuzzing – Lessons Learned

- **Scalability**: Automation is easier than manual efforts (even if limited)

→ **Extensible**: Able to scan way more variants (often minor changes)

CPU Fuzzing – Lessons Learned

- **Scalability**: Automation is easier than manual efforts (even if limited)

→ **Extensible**: Able to scan way more variants (often minor changes)

- **Versatile**: Execute on various microarchitectures

Open-Source – Feel Free to Test

All our tools are **accessible** and **open source**:

Open-Source – Feel Free to Test

All our tools are **accessible** and **open source**:

→ `https://github.com/D4nielMoghimi/medusa`

→ `https://github.com/CISPA/osiris`

→ `https://github.com/IAIK/msrevelio`

Takeaways

- Automated approaches are a **fruitful** way in

Takeaways

- Automated approaches are a **fruitful** way in
  - finding **new and unknown** attack variants

Takeaways

- Automated approaches are a **fruitful** way in
  - finding **new and unknown** attack variants
  - **regression** testing

Takeaways

- Automated approaches are a **fruitful** way in
  - finding **new and unknown** attack variants
  - **regression** testing
  - **extensible** for new building blocks and architectures

Takeaways

- Automated approaches are a **fruitful** way in
  - finding **new and unknown** attack variants
  - **regression** testing
  - **extensible** for new building blocks and architectures
- Despite **limitations**, great toolkit and support for manual efforts

Takeaways

- Automated approaches are a **fruitful** way in
  - finding **new and unknown** attack variants
  - **regression** testing
  - **extensible** for new building blocks and architectures
- Despite **limitations**, great toolkit and support for manual efforts
- Play an **essential role** in CPU research

CISPA
HELMHOLTZ CENTER FOR INFORMATION SECURITY

CPU Fuzzing

Automatic Discovery of Microarchitectural Attacks

Daniel Weber, Michael Schwarz | May 12, 2023

References

- Slides partially by Moritz Lipp
- Icons and Images from `storyset.com`
- Images from CNN