Osiris: Automated Discovery of Microarchitectural Side Channels

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

August 2021

CISPA Helmholtz Center for Information Security

Motivation

Microarchitectural timing side channels, e.g., Flush+Reload or Prime+Probe, can be exploited in different scenarios:

- Attack implementations of cryptographic protocols

Motivation

Microarchitectural timing side channels, e.g., Flush+Reload or Prime+Probe, can be exploited in different scenarios:

- Attack implementations of cryptographic protocols
- As building blocks for transient-execution attacks

Motivation

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Microarchitectural timing side channels, e.g., Flush+Reload or Prime+Probe, can be exploited in different scenarios:

- Attack implementations of cryptographic protocols
- As building blocks for transient-execution attacks

Finding side channels is a **labour-intensive** and **time-consuming** process

Motivation

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Microarchitectural timing side channels, e.g., Flush+Reload or Prime+Probe, can be exploited in different scenarios:

- Attack implementations of cryptographic protocols
- As building blocks for transient-execution attacks

Finding side channels is a **labour-intensive** and **time-consuming** process

We **improve** this process **by fuzzing** for side channels!

Flush+Reload

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)



// potentially executed victim code
*probe_memory;



before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)
```

```
// potentially executed victim code
*probe_memory;
```

```
before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)



// potentially executed victim code
*probe_memory;



before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)



// potentially executed victim code
*probe_memory;



before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload

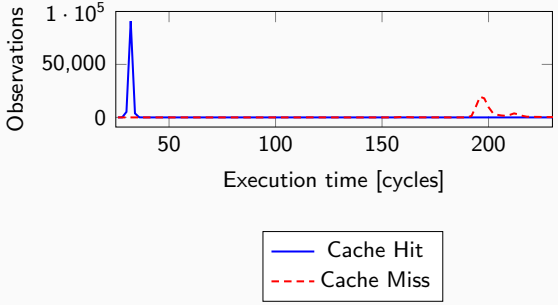
```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)



// potentially executed victim code
*probe_memory;



before = time();
*probe_memory;
after = time();
plot(after-before);
```



Flush+Reload – Sequence Triple

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)
```

Reset Sequence

```
// potentially executed victim code
*probe_memory;
```

```
before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload – Sequence Triple

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)
```

Reset Sequence

```
// potentially executed victim code
*probe_memory;
```

Trigger Sequence

```
before = time();
*probe_memory;
after = time();
plot(after-before);
```

Flush+Reload – Sequence Triple

```
int* probe_memory = malloc(4096);
_mm_clflush(probe_memory)
```

Reset Sequence

```
// potentially executed victim code
*probe_memory;
```

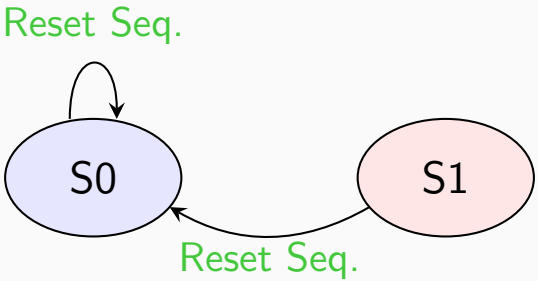
Trigger Sequence

```
before = time();
*probe_memory;
after = time();
plot(after-before);
```

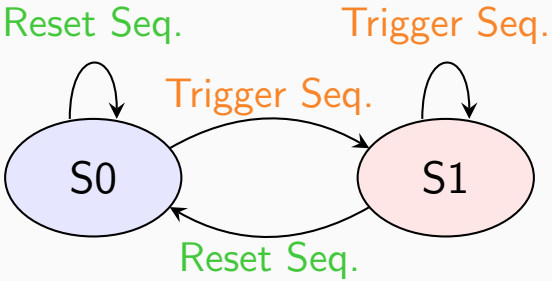
Measurement Sequence

Sequence Triples

Sequence Triples

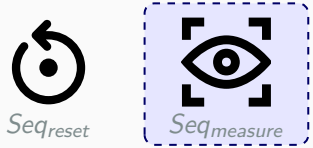


Sequence Triples



Testing A Sequence Triple

Testing A Sequence Triple



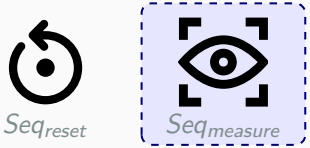
Testing A Sequence Triple

Seq_{reset}

$Seq_{measure}$

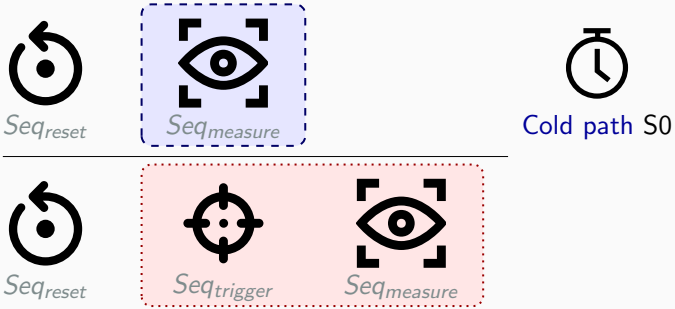
Cold path S0

Testing A Sequence Triple

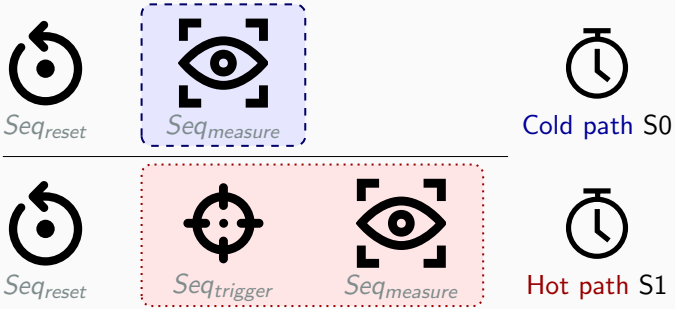


Cold path S0

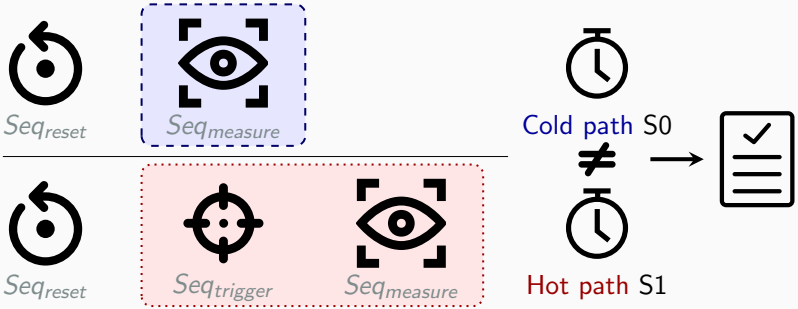
Testing A Sequence Triple



Testing A Sequence Triple

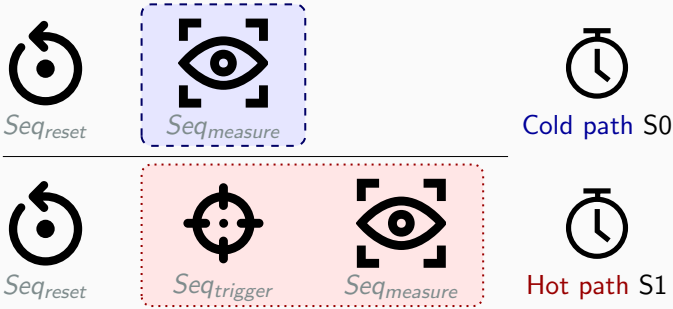


Testing A Sequence Triple



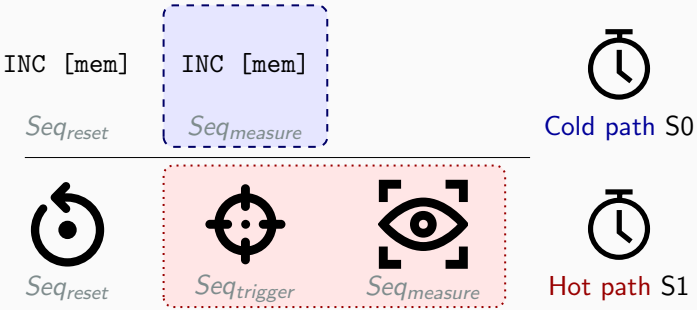
Testing A Sequence Triple

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



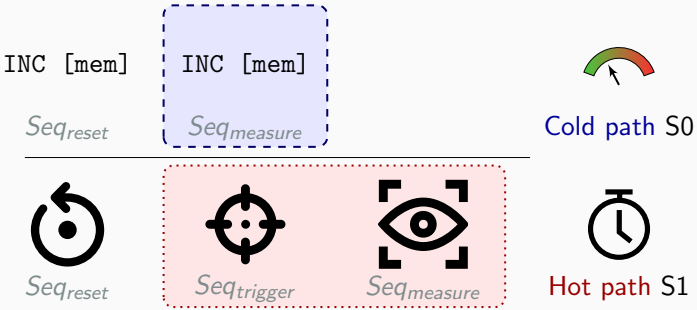
Testing A Sequence Triple

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



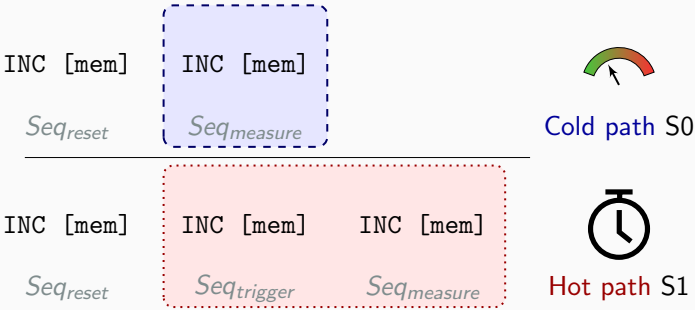
Testing A Sequence Triple

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



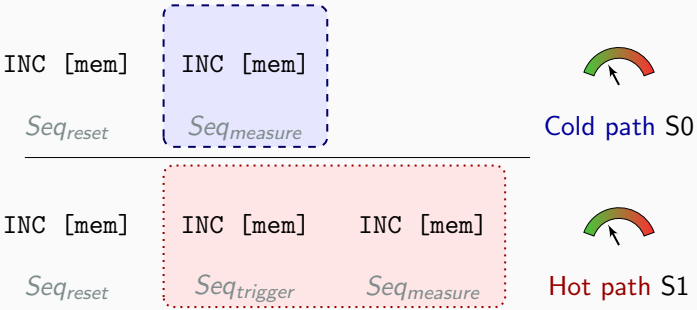
Testing A Sequence Triple

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



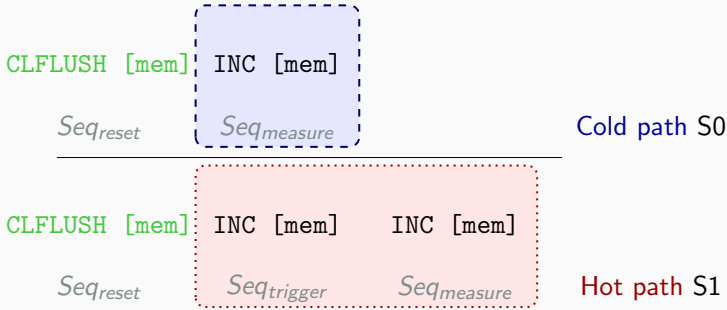
Testing A Sequence Triple

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



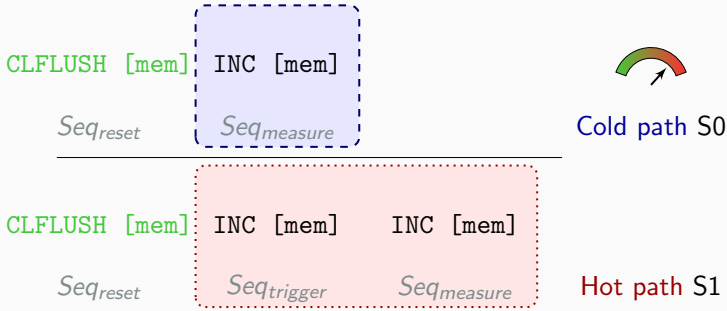
Testing A Sequence Triple

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



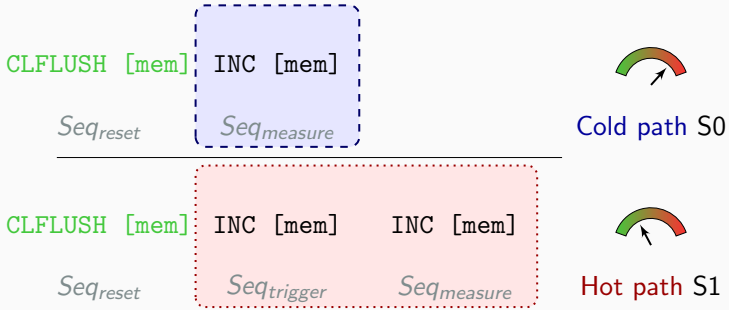
Testing A Sequence Triple

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



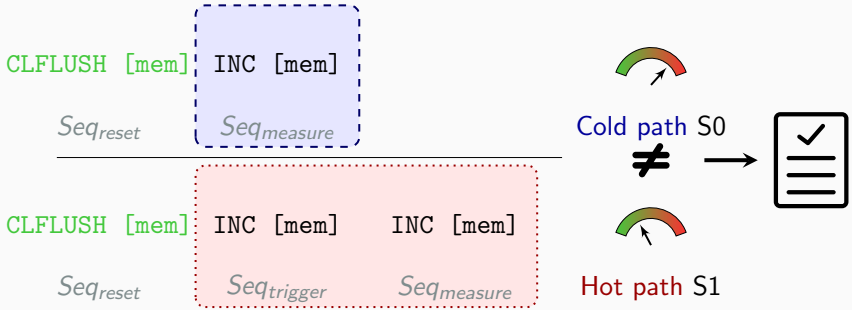
Testing A Sequence Triple

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

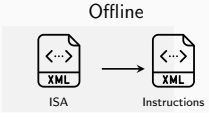


Testing A Sequence Triple

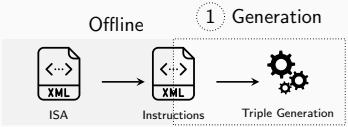
Example 2: $Seq_{measure}$ = $Seq_{trigger}$ = `INC [mem]`;
Seq_{reset} = `CLFLUSH [mem]`



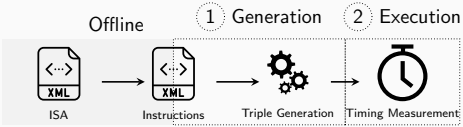
Osiris – Fuzzing x86 CPUs for Side Channels



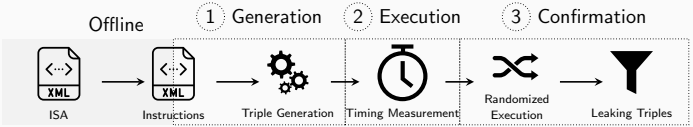
Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels

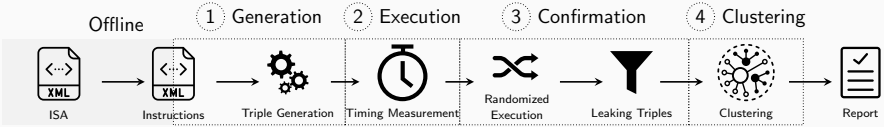


Osiris – Fuzzing x86 CPUs for Side Channels

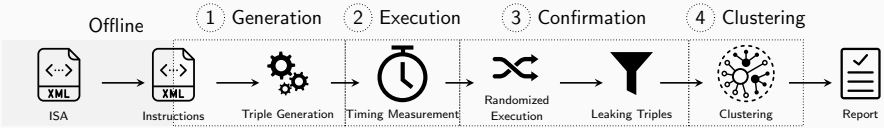


Osiris – Fuzzing x86 CPUs for Side Channels

Osiris – Fuzzing x86 CPUs for Side Channels



Osiris – Fuzzing x86 CPUs for Side Channels



We tested Osiris on **5 different CPUs** (AMD and Intel)

Fuzzing Results

- Runtime of up to $\sim$ 4 days per CPU

Fuzzing Results

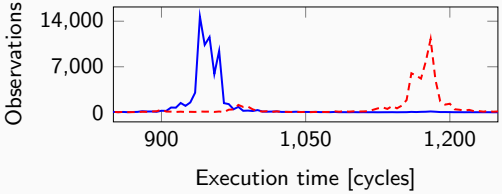
- Runtime of up to $\sim$ 4 days per CPU
- $<$ 30 clusters per CPU

Fuzzing Results

- Runtime of up to $\sim$ 4 days per CPU
- $<$ 30 clusters per CPU
- Found different known and 4 novel side channels

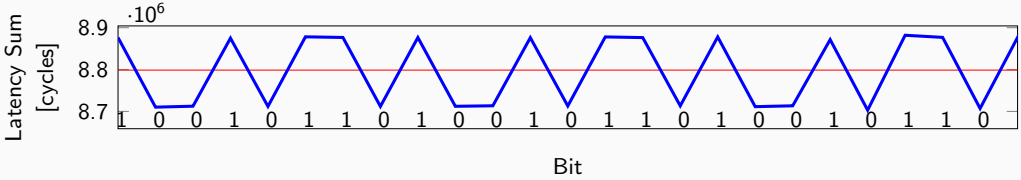
Fuzzing Results

- Runtime of up to $\sim$ 4 days per CPU
- $<$ 30 clusters per CPU
- Found different known and 4 novel side channels



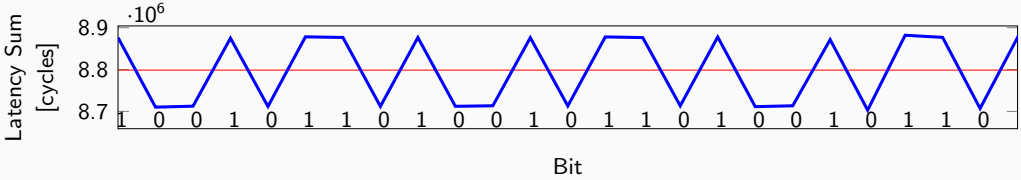
RDRAND-Based Side channel

Case Studies – RDRAND Covert Channel



RDRAND Covert Channel Transmission on Ice-Lake

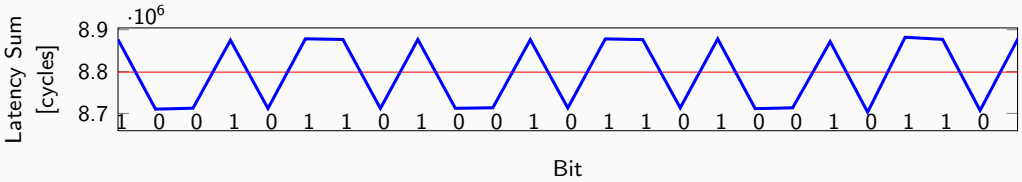
Case Studies – RDRAND Covert Channel



RDRAND Covert Channel Transmission on Ice-Lake

(works **cross-core cross-VM**)

Case Studies – RDRAND Covert Channel



RDRAND Covert Channel Transmission on Ice-Lake

(works **cross-core cross-VM**)

Few requirements: no shared memory, no shared cache

Stealthy: no related performance counters

Improving Transient-Execution Attacks

- **Mounted Spectre and Meltdown** attacks using the novel side channels

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Improving Transient-Execution Attacks

- **Mounted Spectre and Meltdown** attacks using the novel side channels
- Spectre PoC was **2.4x as fast** as with a similar non-cache side channel

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Improving Transient-Execution Attacks

- **Mounted Spectre and Meltdown** attacks using the novel side channels
- Spectre PoC was **2.4x as fast** as with a similar non-cache side channel
- Meltdown PoC leaked **7.83 bytes at once** (previously 3 bytes)

Case Studies – Improving Attacks and KASLR break

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Improving Transient-Execution Attacks

- **Mounted Spectre and Meltdown** attacks using the novel side channels
- Spectre PoC was **2.4x as fast** as with a similar non-cache side channel
- Meltdown PoC leaked **7.83 bytes at once** (previously 3 bytes)

FlushConflict

- **KASLR break** based on transient execution

Case Studies – Improving Attacks and KASLR break

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Improving Transient-Execution Attacks

- **Mounted Spectre and Meltdown** attacks using the novel side channels
- Spectre PoC was **2.4x as fast** as with a similar non-cache side channel
- Meltdown PoC leaked **7.83 bytes at once** (previously 3 bytes)

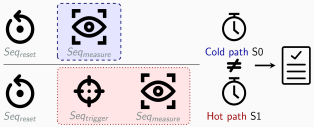
FlushConflict

- **KASLR break** based on transient execution
- Works **on new Intel CPUs** (Ice Lake and Comet Lake)

Summary

Osiris: Automated Discovery of Microarchitectural Side Channels

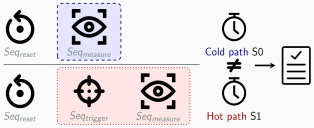
- We denote **side channels as sequence triples**



Summary

Osiris: Automated Discovery of Microarchitectural Side Channels

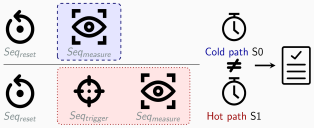
- We denote **side channels as sequence triples**
- Osiris **fuzzes for CPU timing side channels**



Summary

Osiris: Automated Discovery of Microarchitectural Side Channels

- We denote **side channels as sequence triples**
- Osiris **fuzzes for CPU timing side channels**
- We found **4 novel side channels**

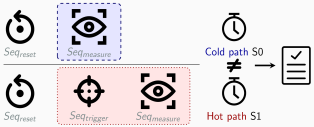


Summary

Osiris: Automated Discovery of Microarchitectural Side Channels

- We denote **side channels as sequence triples**
- Osiris **fuzzes for CPU timing side channels**

- We found **4 novel side channels**
- We evaluated these side channels in **3 case studies**



Summary

Osiris: Automated Discovery of Microarchitectural Side Channels

- We denote **side channels as sequence triples**
- Osiris **fuzzes for CPU timing side channels**

- We found **4 novel side channels**
- We evaluated these side channels in **3 case studies**

- We **open-sourced Osiris** on `https://github.com/cispa/osiris`