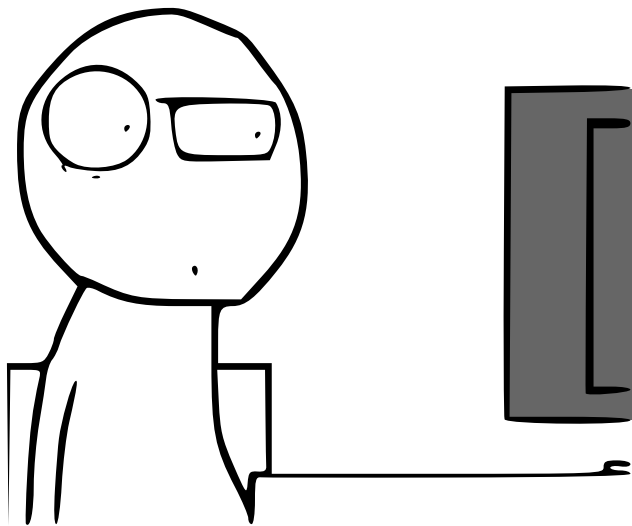
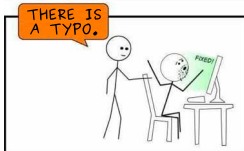
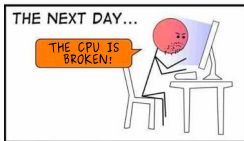
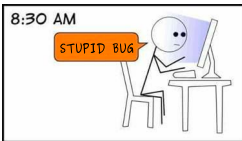


CPU Vulnerabilities

Finding Bugs Where Software Ends

Michael Schwarz | FOSAD Summer School 2026







Who am I

Research on CPU security and
side channels

✉ michael.schwarz@cispa.de



Dr. Michael Schwarz

Faculty @ CISPA



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



ÆPIC Leak

*Architectural leaks
pioneer*



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



ÆPIC Leak

*Architectural leaks
pioneer*



PLATYPUS

*Software power side
channels*



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



ÆPIC Leak

*Architectural leaks
pioneer*



PLATYPUS

*Software power side
channels*

**ZombieLoad, CacheWarp, GhostWrite,
StackWarp, LoongLeak, and more**



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



ÆPIC Leak

*Architectural leaks
pioneer*



PLATYPUS

*Software power side
channels*



**ZombieLoad, CacheWarp, GhostWrite,
StackWarp, LoongLeak, and more**

KPTI, Cloudflare

Defenses in production



Why Should We Listen to Your Rant?



Spectre, Meltdown

*Transient execution
attacks*



ÆPIC Leak

*Architectural leaks
pioneer*



PLATYPUS

*Software power side
channels*



**ZombieLoad, CacheWarp, GhostWrite,
StackWarp, LoongLeak, and more**

KPTI, Cloudflare

Defenses in production

"Yes, I probably broke your CPU."



CPU's Have Vulnerabilities Too



Meltdown and Spectre



Since Meltdown and Spectre, we know:

CPU's can have
security vulnerabilities
too





Why CPU Vulnerabilities Feel Different

Long Lifetime

Hardware ships before flaws are found.

Big Impact

One core design can appear in many systems.

Repair Friction

Replacement is slower than patching software.

Security consequence

A small mistake can become a platform-wide isolation failure.



The CPU is Critical Infrastructure



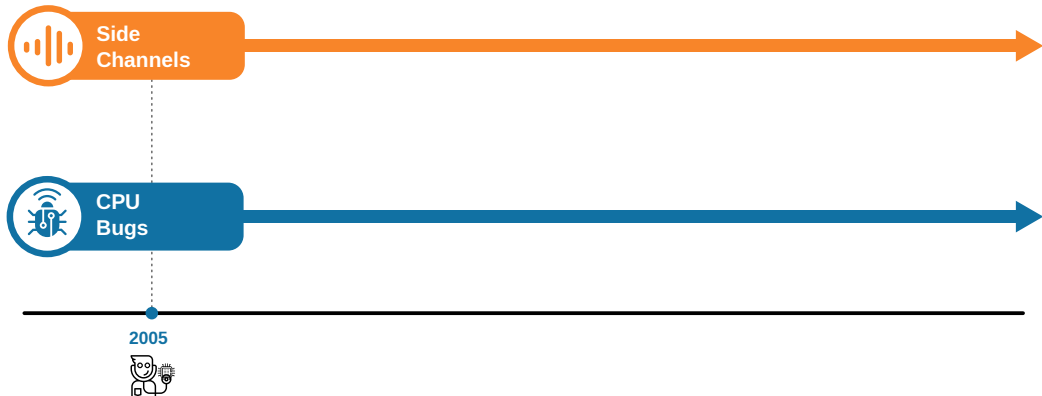
Secure Software needs secure CPUs



**Hospitals, finance, transport, and
clouds all inherit the security
properties of their CPUs**

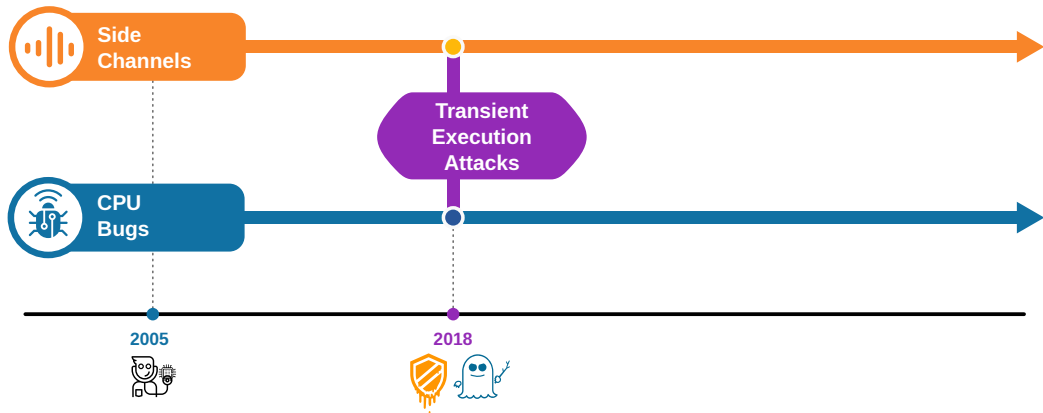


Brief History of CPU Security



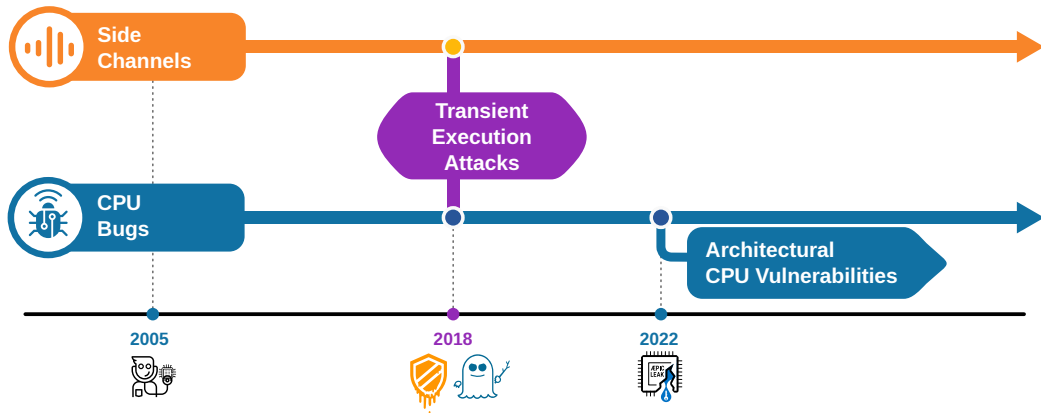


Brief History of CPU Security



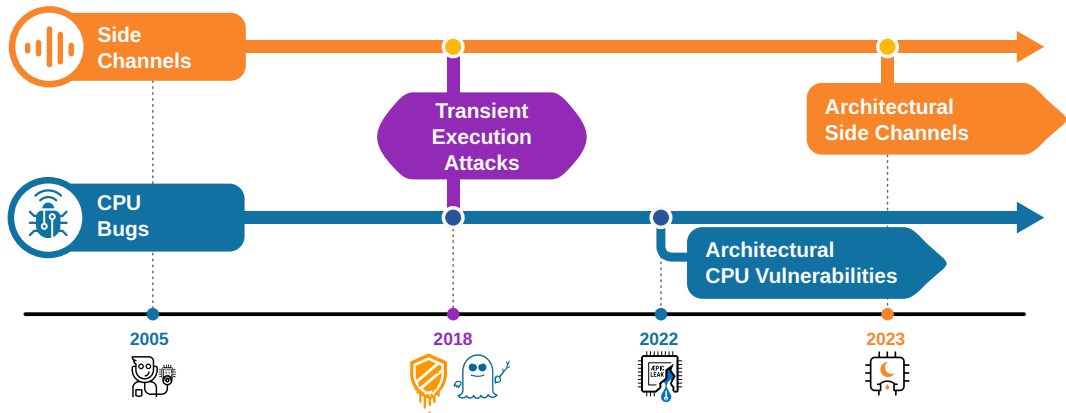


Brief History of CPU Security





Brief History of CPU Security



We focus on **architectural** CPU vulnerabilities



Four hours, two labs

Part I

Architectural meta-data
leakage

Part II

Architectural data
leakage

Both

Crossing privilege
domains to extract
secrets

Takeaway

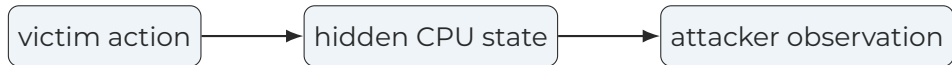
Session 1 starts with **side channels**

Session 2 continues with **direct data leakage**



A motivating question

If a user process obtains the correct architectural result, can it still learn something it was never authorized to know?



The answer is often yes.

Basics





Side Channels in Software



In the physical world, we can
listen for sound.



Side Channels in Software



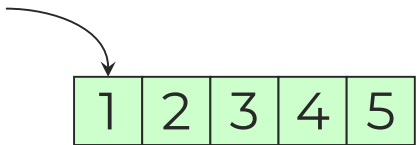
In the physical world, we can
listen for sound.

In software, we listen to
TIME.



Example: PIN Comparison

- Trivial approach: Compare each digit until a difference

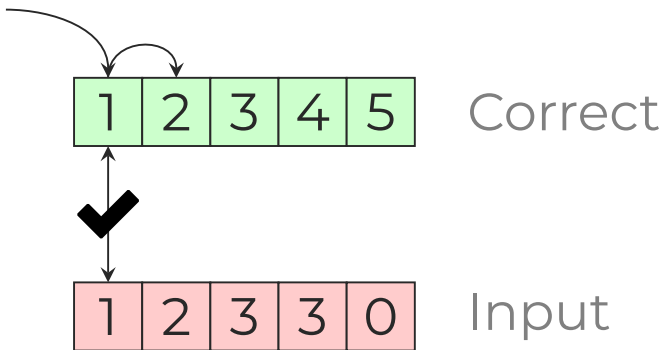
 Correct

 Input



Example: PIN Comparison

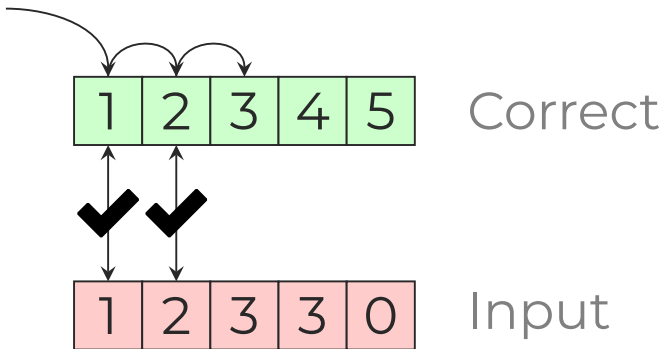
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

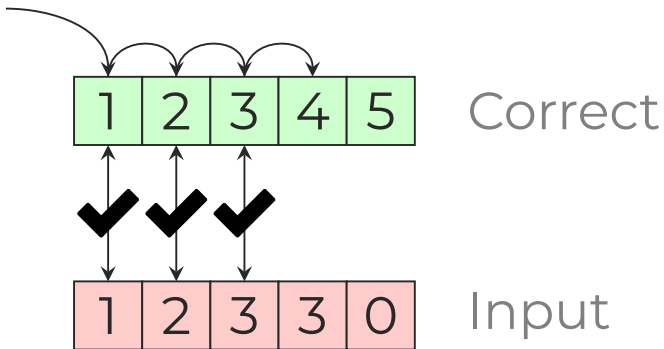
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

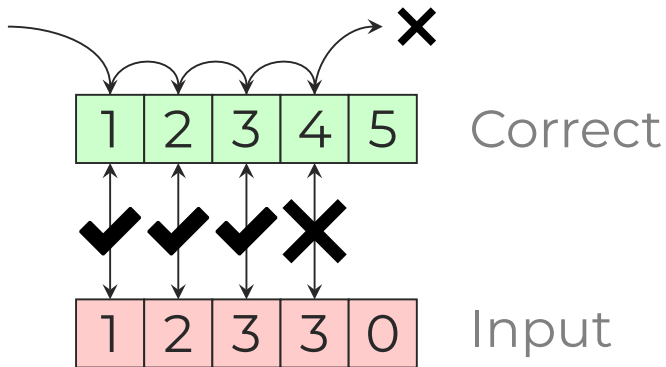
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

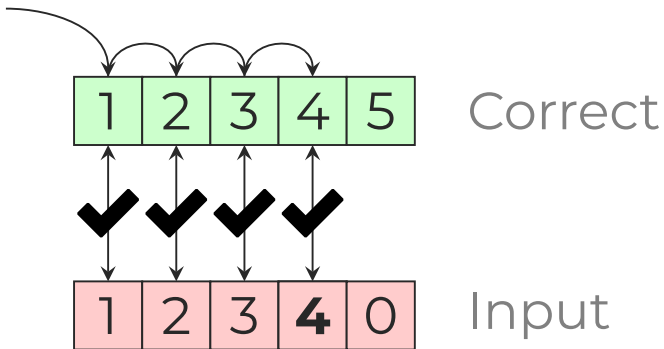
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

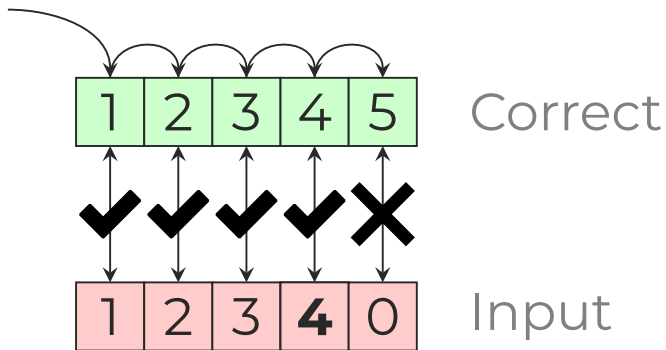
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

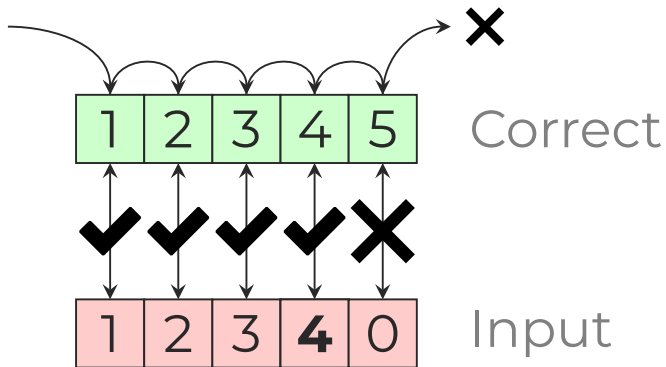
- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

- Trivial approach: Compare each digit until a difference





Example: PIN Comparison

- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	
3000	
...	...

- If digit is **correct**, next digit is checked → **longer** execution time



Example: PIN Comparison

- Measuring the **execution times** for different PINs

PIN	Time
0000	
1000	
2000	
3000	
...	...

- If digit is **correct**, next digit is checked → **longer** execution time
- 10 tries (maximum) to get a digit



Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	
1300	
...	...

- Repeat for every digit



Example: PIN Comparison

- Measuring the execution times for different PINs

PIN	Time
1000	
1100	
1200	
1300	
...	...

- Repeat for every digit
- Longest execution time reveals correct digit



Example: PIN Comparison

One digit at a time: exponential search $\rightarrow$ linear work

Side channel

linear: $10n$



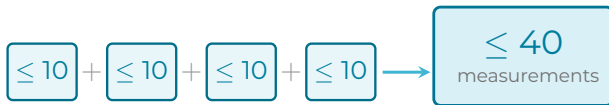


Example: PIN Comparison

One digit at a time: exponential search $\rightarrow$ linear work

Side channel

linear: $10n$



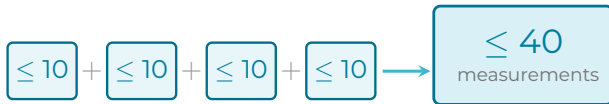


Example: PIN Comparison

One digit at a time: exponential search $\rightarrow$ linear work

Side channel

linear: $10n$



Brute force

exponential: 10^n



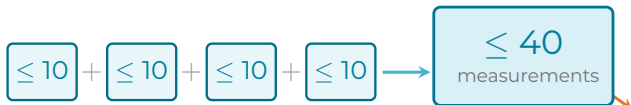


Example: PIN Comparison

One digit at a time: exponential search $\rightarrow$ linear work

Side channel

linear: $10n$



Brute force

exponential: 10^n

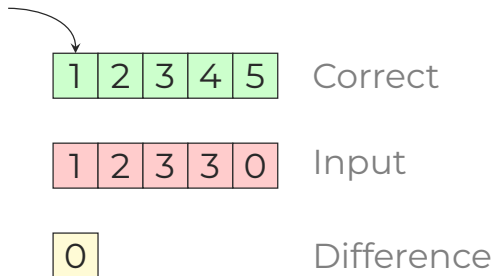


250×
fewer tries



Example: PIN Comparison

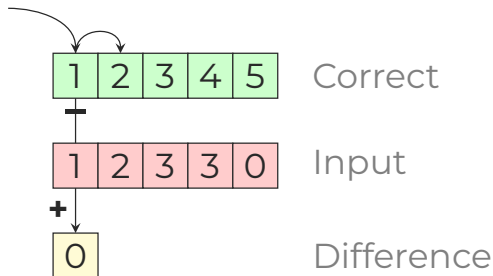
- Many functions can be implemented with [secret-independent runtime](#)





Example: PIN Comparison

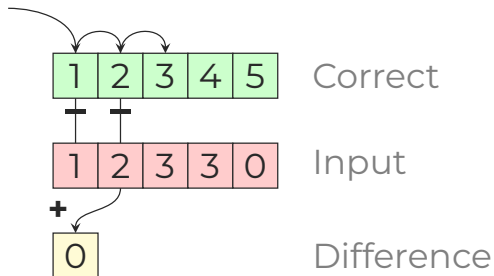
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

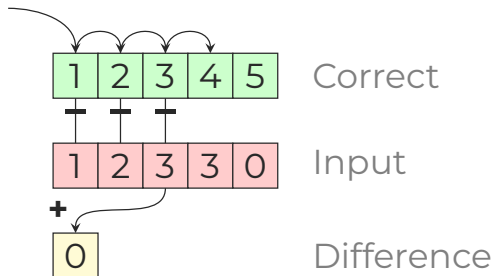
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

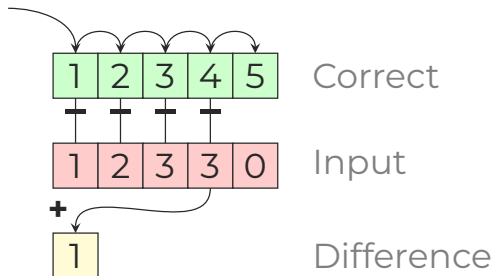
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

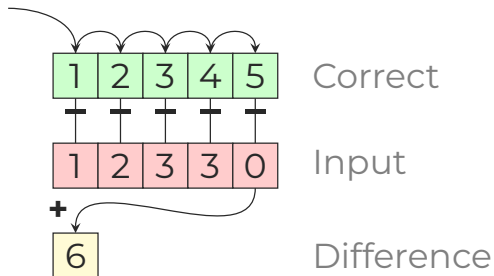
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

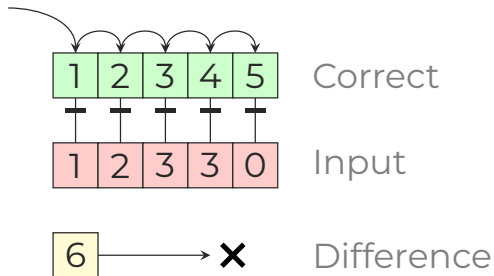
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

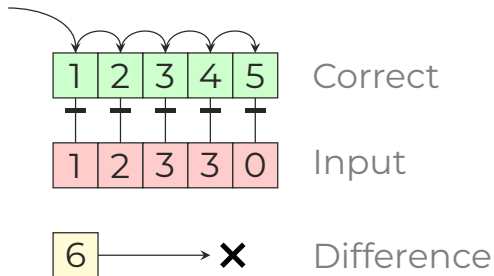
- Many functions can be implemented with **secret-independent runtime**





Example: PIN Comparison

- Many functions can be implemented with **secret-independent runtime**



- Sometimes it is the **hardware** that leaks



What To Do With a Side Channel?



Direct attacks (mostly crypto)



Fingerprinting attack targets



Exploit helper



Building block for CPU bugs

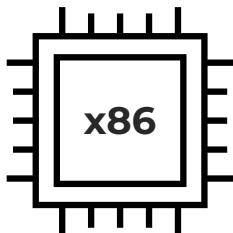
**“I doubt you can measure
these small difference in
practice”**



Observing a CPU

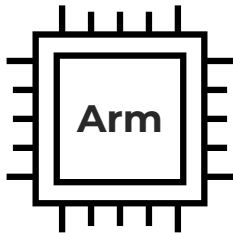


High-resolution Timers



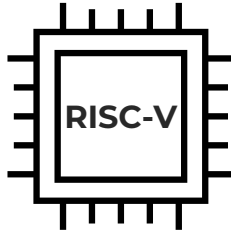
`rdtsc`

CPU cycles, unprivileged



`PMCCNTR_ELO`

CPU cycles, sometimes unprivileged



`rdcycle`

CPU cycles, sometimes unprivileged



Measuring Time

$$x = y + 1$$



Measuring Time

start = 

x = y + 1

end = 



Measuring Time

```
start = 
```

```
x = y + 1
```

```
end = 
```

```
 $\Delta$  = end - start
```



Measuring Time

```
start = 
```

```
x = y + 1
```

```
end = 
```

```
 $\Delta$  = end - start
```

1. run: Δ = 302



Measuring Time

```
start = 
```

```
x = y + 1
```

```
end = 
```

```
 $\Delta$  = end - start
```

1. run: Δ = 302

2. run: Δ = 54



Measuring Time

```
start = 
```

```
x = y + 1
```

```
end = 
```

```
 $\Delta$  = end - start
```

1. run: Δ = 302

2. run: Δ = 54

Microarchitectural State

Access $\rightarrow$ cache $\rightarrow$ faster execution. Well understood and **expected**.

Time + Threshold

Time + Threshold

Time + Threshold

```
reload(addr):  
    t0 = time()  
    load(addr)  
    t1 = time()  
    return t1-t0 < threshold
```

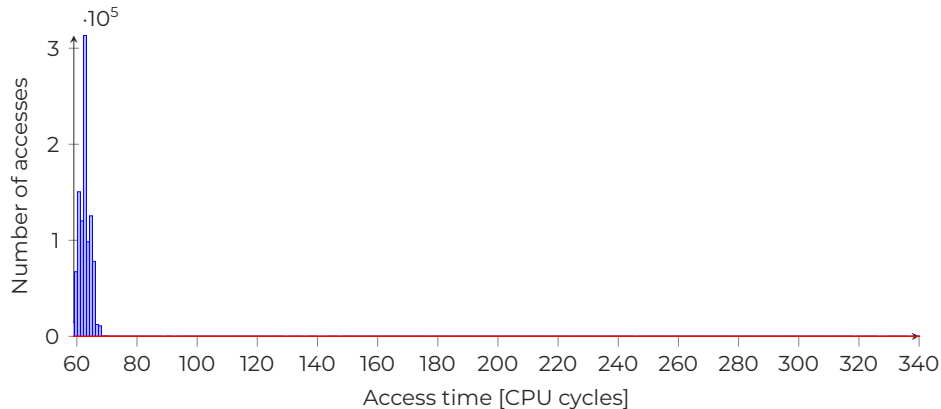
Time + Threshold

```
reload(addr):  
    t0 = time()  
    load(addr)  
    t1 = time()  
    return t1-t0 < threshold
```



Cached vs Uncached

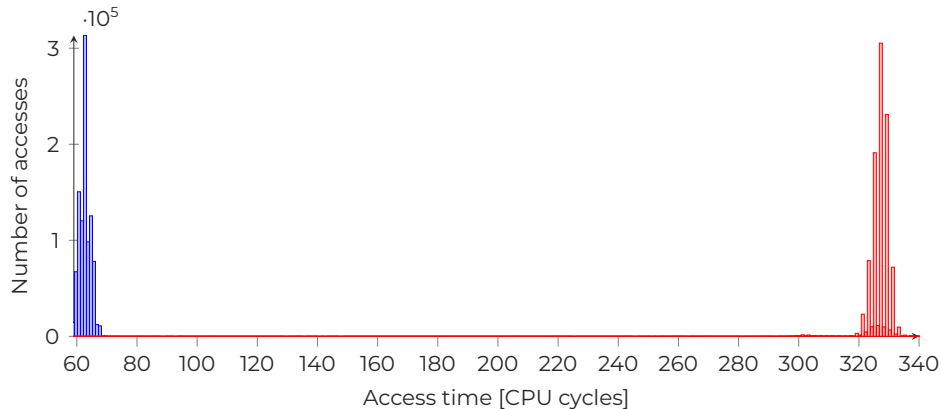
Cache Hits





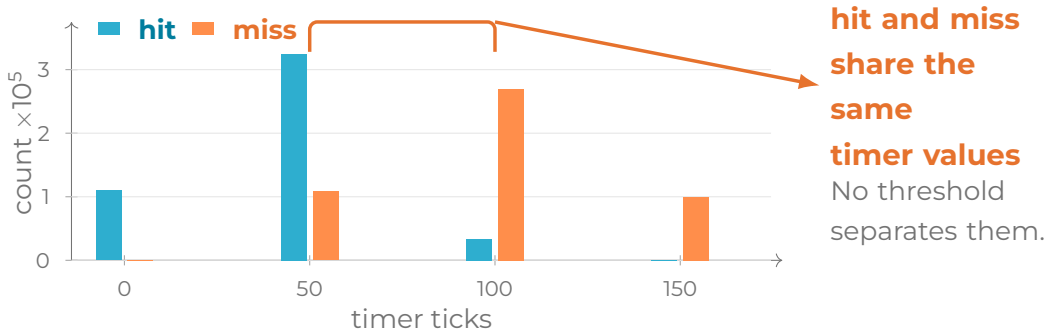
Cached vs Uncached

Cache Hits Cache Misses





Timer Resolution



Takeaway

Low timer resolution $\rightarrow$ no useful architectural signal

To offer protection against timing attacks and [fingerprinting](#), `performance.now()` is coarsened based on whether or not the document is [cross-origin isolated](#).

- Resolution in isolated contexts: 5 microseconds
- Resolution in non-isolated contexts: 100 microseconds

Before version 91, [timer resolutions](#) in Chrome were restricted to 5 microseconds on desktop, where [site-isolation](#) is enabled, and to 100 microseconds on Android, where it's not.

Starting from version 91, following a [specification change](#), Chrome will be restricting the resolution of explicit [timers](#) (`performance.now()`, `performance.timeOrigin`, and other performance APIs that expose `DOMHighResTimestamps`) to 100 microseconds across platforms. By enabling [cross-origin isolation](#), websites can relax the restriction to 5 microseconds regardless of platform.

Commit 25e5753

 **rniwa** committed on Jan 8, 2018

Reduce the precision of "high" resolution time to 1ms

https://bugs.webkit.org/show_bug.cgi?id=189910

<rdar://problem/36865943>

Reviewed by Saam Barati.

Source/WebCore:

Reduced the high prevision time's resolution to 1ms, the same precision as `Date.now()`.

To offer protection against timing attacks and [fingerprinting](#), `performance.now()` is coarsened based on whether or not the document is [cross-origin isolated](#).

- Resolution in isolated contexts: 5 microseconds
- Resolution in non-isolated contexts: 100 microseconds

Before version 91, [timer resolutions](#) in Chrome were restricted to 5 microseconds on desktop, where [site-isolation](#) is enabled, and to 100 microseconds on Android, where it's not.

Starting from version 91, following a [specification change](#), Chrome will be restricting the resolution of explicit [timers](#) (`performance.now()`, `performance.timeOrigin`, and other performance APIs that expose `DOMHighResTimestamps`) to 100 microseconds across platforms. By enabling [cross-origin isolation](#), websites can relax the restriction to 5 microseconds regardless of platform.

Commit 25e5753

 rnlwa committed on Jan 8, 2018

Reduce the precision of "high" resolution time to 1ms

https://bugs.webkit.org/show_bug.cgi?id=189910

<rdar://problem/36865943>

Reviewed by Saam Barati.

Source/WebCore:

Reduced the high precision time's resolution to 1ms, the same precision as `Date.now()`.

What if we have
no timer at all?

Time + Threshold

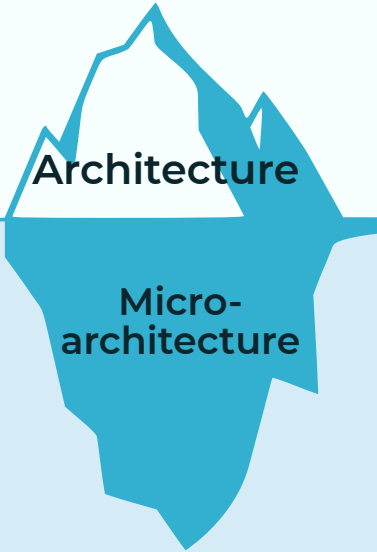
```
reload(addr):  
    t0 = time()  
    load(addr)  
    t1 = time()  
    return t1-t0 < threshold
```

Time + Threshold

```
reload(addr):  
    t0 = time()  
    load(addr)  
    t1 = time()  
    return t1-t0 < threshold
```



```
reload_oracle(addr):  
    if addr in cache:  
        return 1  
    return 0
```



Architecture

The diagram features a blue iceberg floating on a light blue background representing water. A horizontal line divides the iceberg into two parts. The smaller, visible tip above the line is labeled 'Architecture'. The much larger, submerged part below the line is labeled 'Micro-architecture'. The iceberg has a jagged, mountain-like top.

**Micro-
architecture**



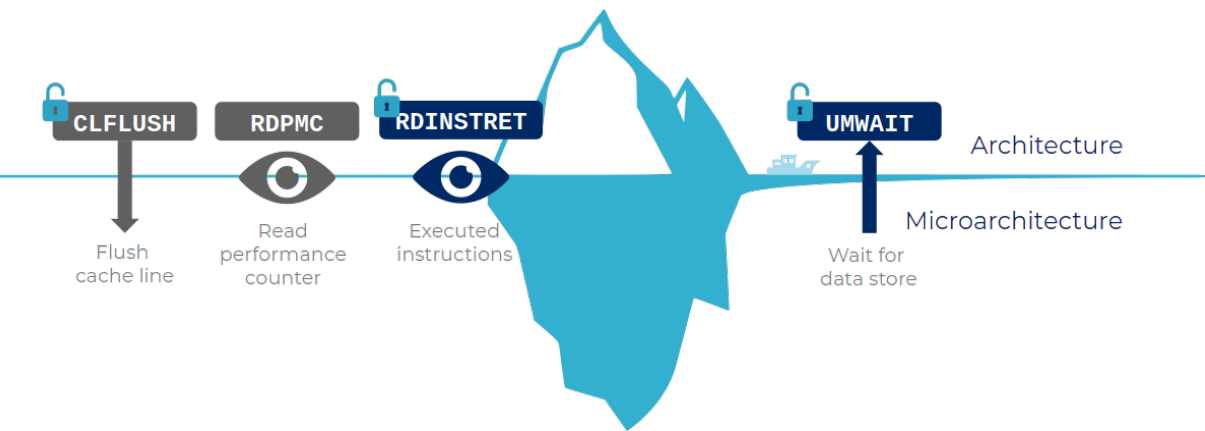
Architectural versus Microarchitectural

Architectural

- Register value
- Value in memory
- CPU flag
- Exception

Microarchitectural

- State of internal element (e.g., cache)
- Runtime differences
- Power consumption
- Stalls and contention



possible bases



boot chooses one random offset



this boot



secret base



KASLR Break

Kernel

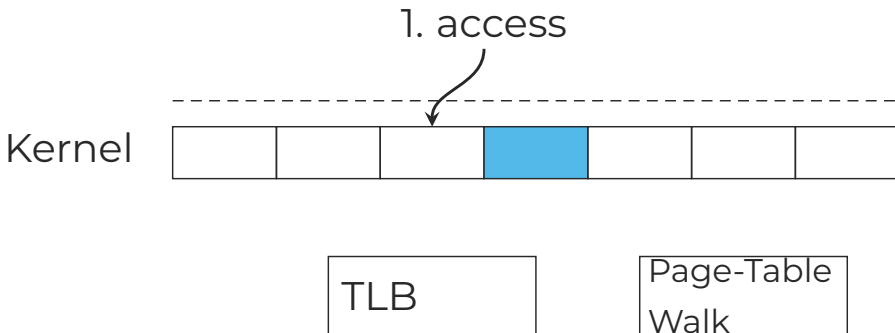


TLB

Page-Table
Walk

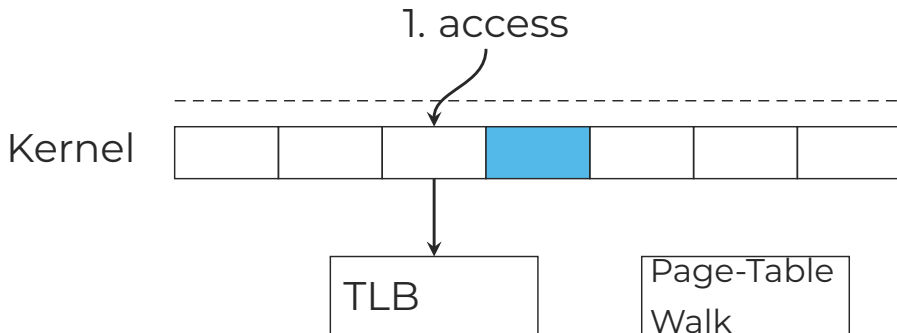


KASLR Break



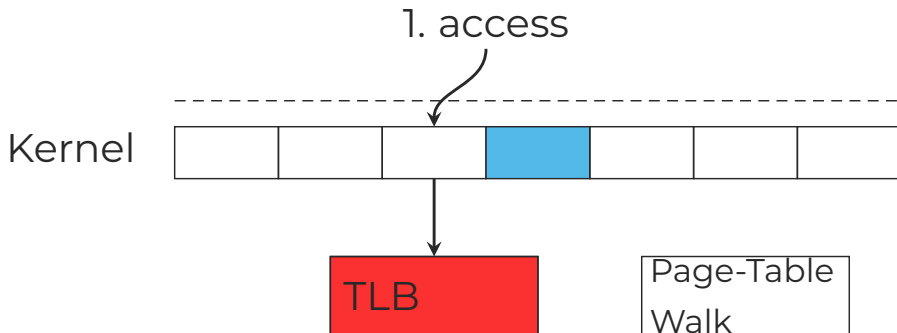


KASLR Break



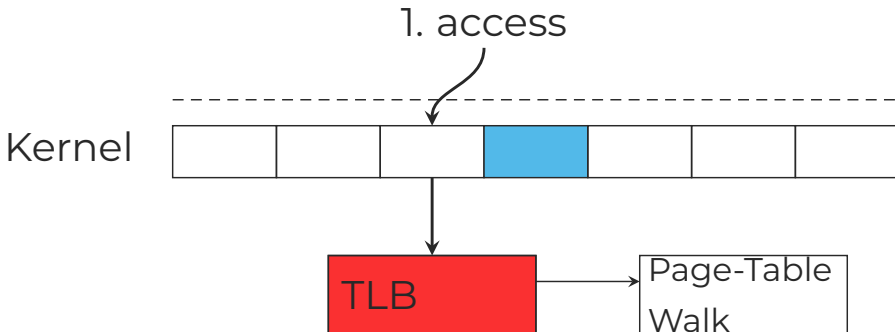


KASLR Break



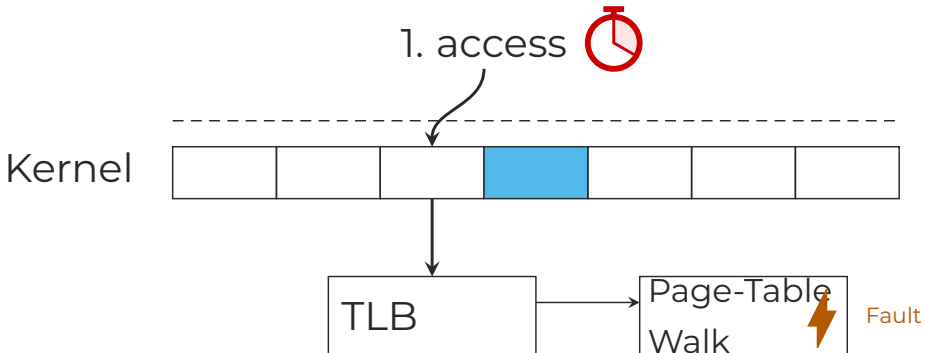


KASLR Break



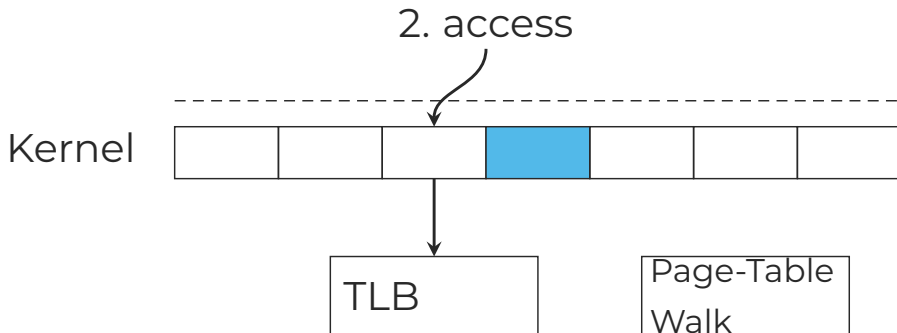


KASLR Break



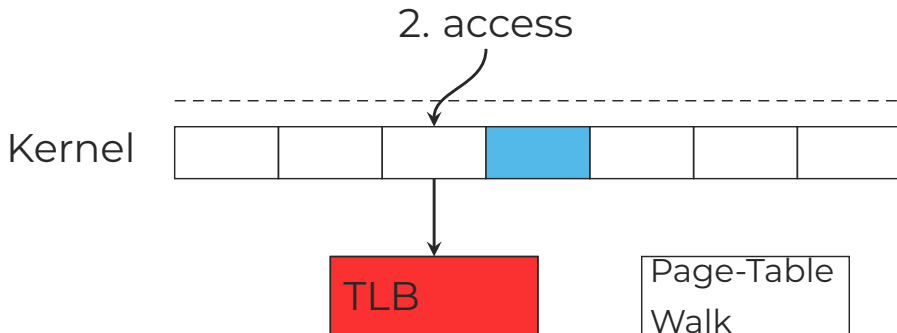


KASLR Break



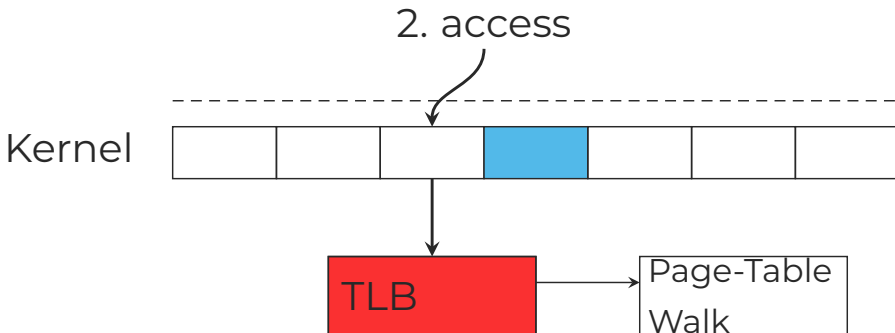


KASLR Break



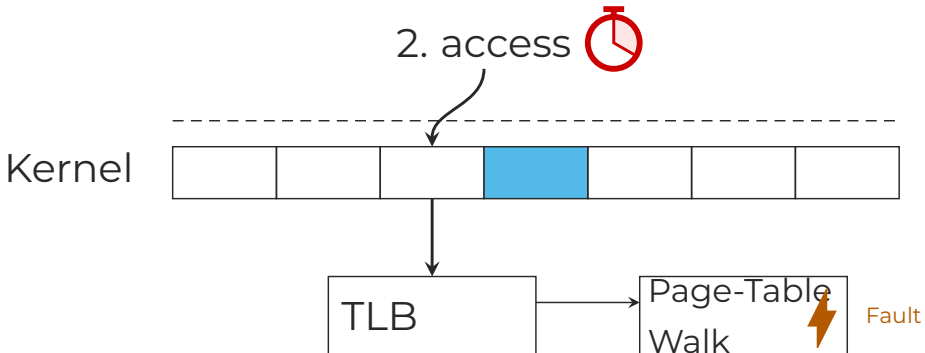


KASLR Break



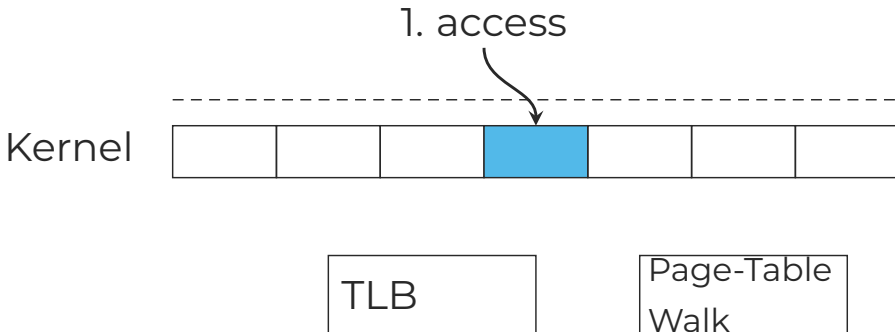


KASLR Break



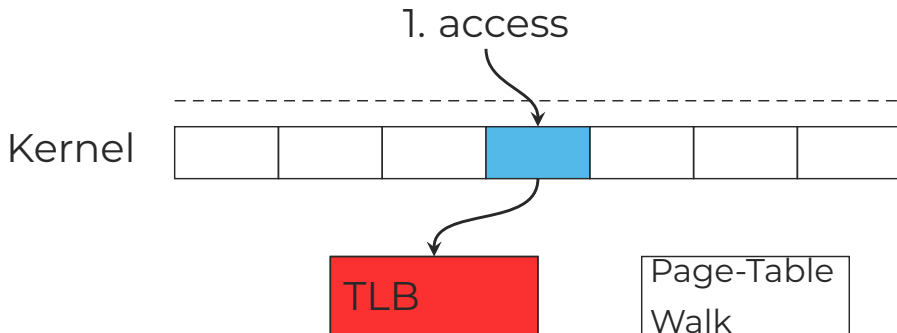


KASLR Break



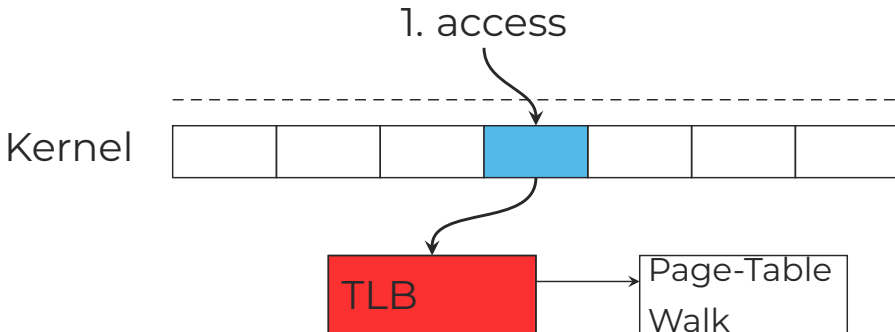


KASLR Break



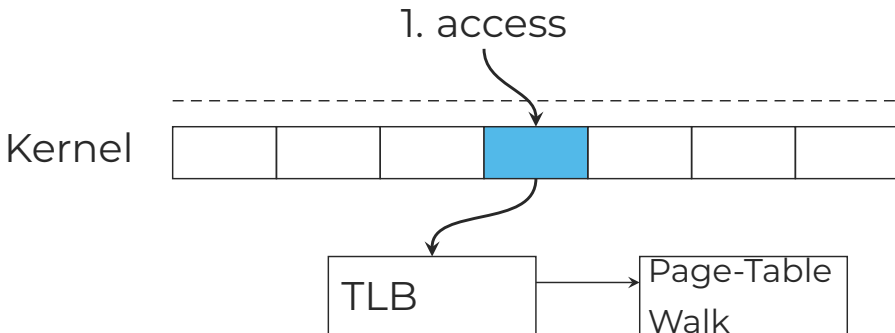


KASLR Break



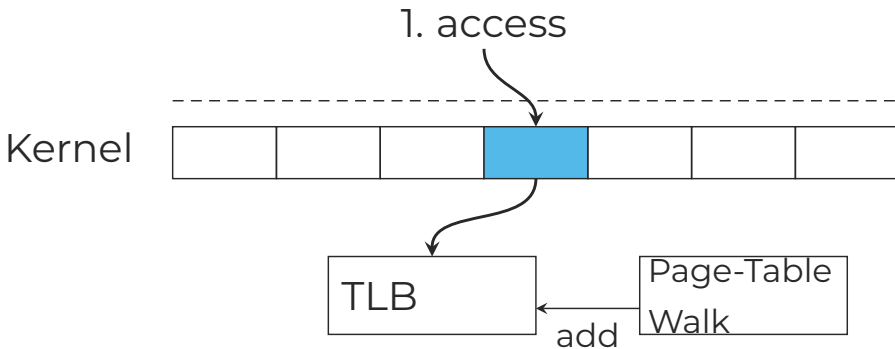


KASLR Break



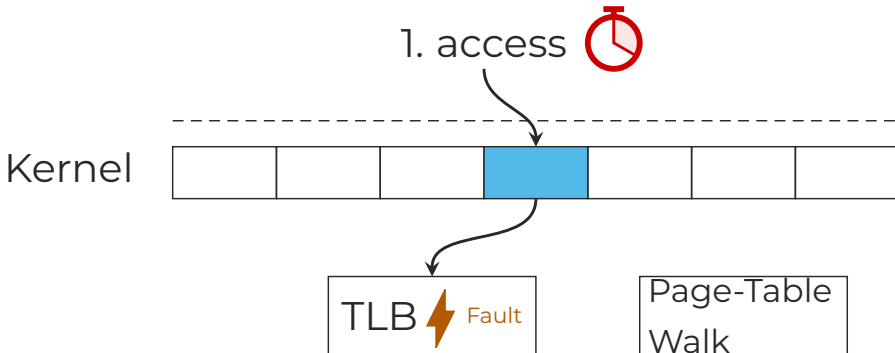


KASLR Break



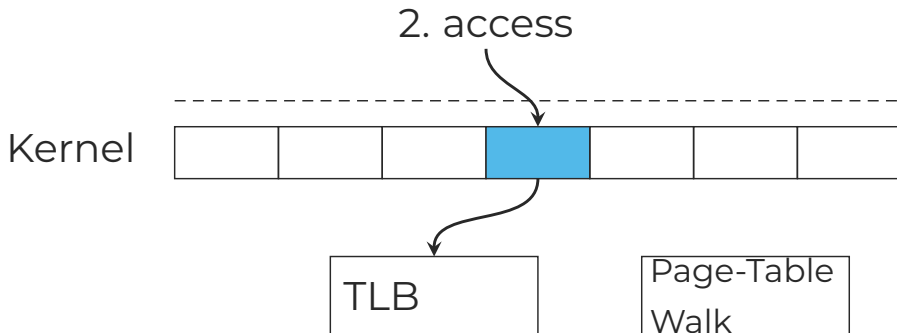


KASLR Break



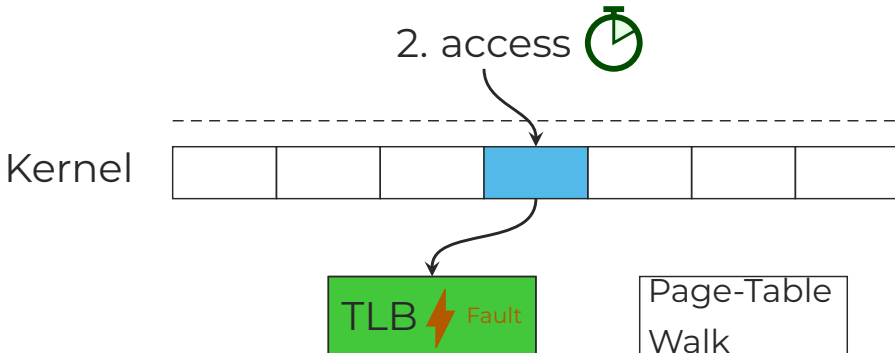


KASLR Break



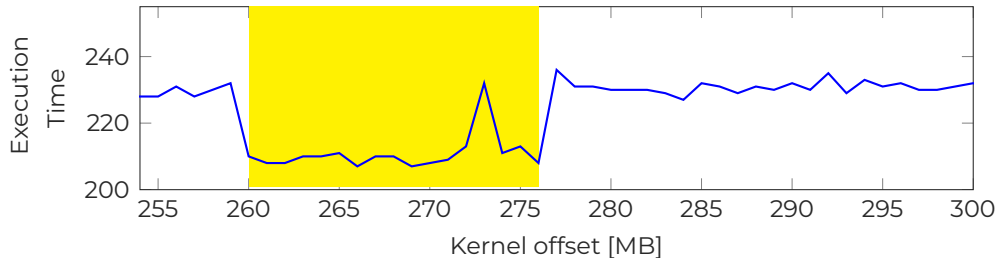


KASLR Break





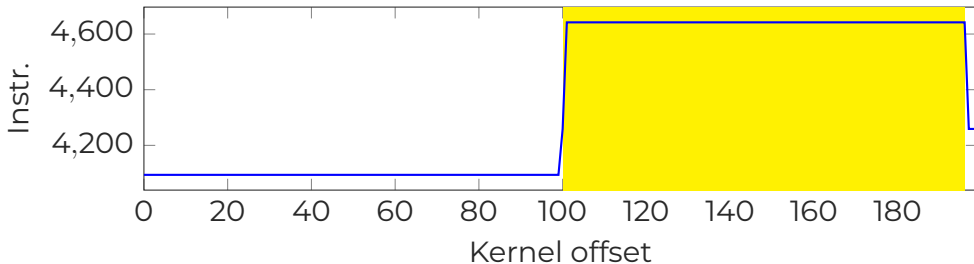
KASLR Break - Timing



x86 (AMD)



KASLR Break - Retired Instructions



RISC-V (U74)

**“Nice example, but this is not
generic”**



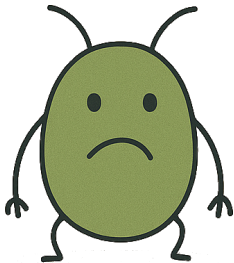
Finding Vulnerabilities



CPU Security Testing is in its Infancy

Technique	Software	CPU
Manual	✓	✓
Random Fuzzing	✓	✓
Guided Fuzzing	✓	~
Static Analysis	✓	~
Symbolic Execution	✓	~
Sanitizer	✓	✗

Software Bug

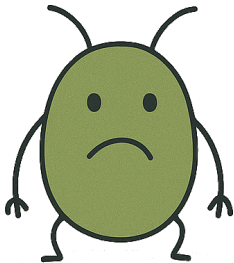


Often just crashes

Directly seen

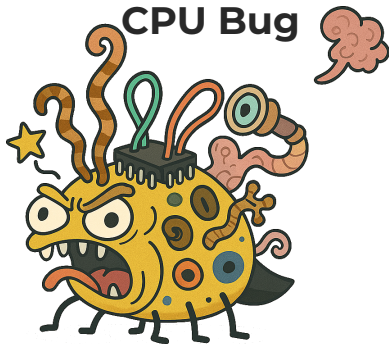
Per program

Software Bug



Often just crashes
Directly seen
Per program

CPU Bug



Refuses to crash
Indirectly seen
Complex state and interaction



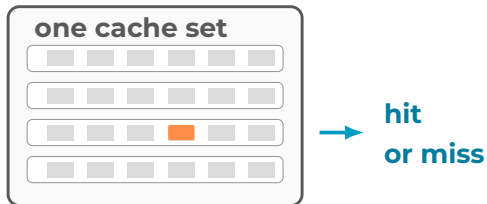
Software Testing vs CPU Testing

Feature	Software	CPU
Input	Data	Code
Can run in a debugger	✓	~
Has feedback	✓	~
Emulation	fast	slow
Crashes	✓	rarely
Execution	linear	parallel
Has laws of physics	✗	✓



Start With a Target

Target: the cache



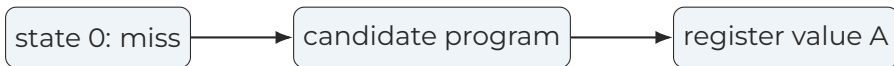
- Two easy states: **cached** or **not cached**
- Software can prepare either state
- Victim activity changes the state

Takeaway

Can an instruction sequence convert **cache state** into a normal **architectural value**?



A Simple Oracle Shape



Repeat with state 1 and look for register value B

Takeaway

The observation must be reliable enough to distinguish distributions



ExfilState: Finding a Leak

- Generate random instruction sequence

```
LDXR    x2, [x0]  
STXR    x3, x4, [x0]
```



ExfilState: Finding a Leak

cache

[x0]

[x1]

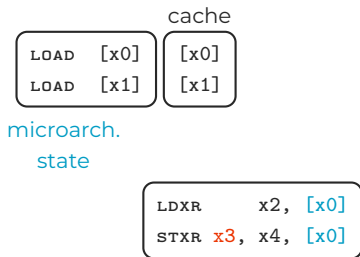
LDXR x2, [x0]

STXR x3, x4, [x0]

- Generate random instruction sequence
- Generate cache state



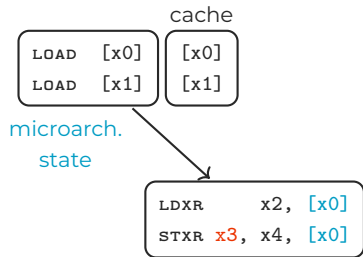
ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime **microarchitectural state**



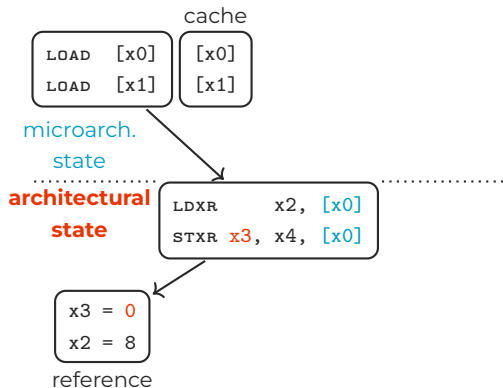
ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime **microarchitectural state**
- Run sequence



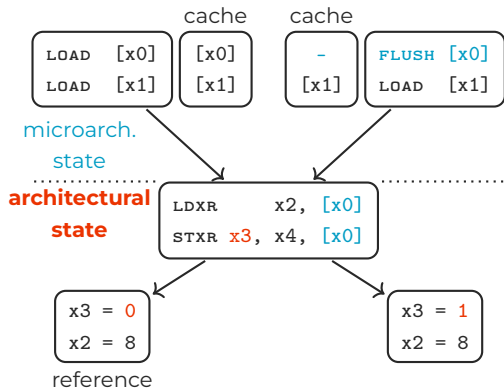
ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime **microarchitectural state**
- Run sequence
- Collect **architectural state**



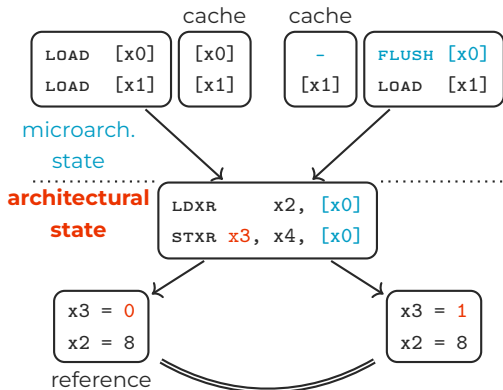
ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime **microarchitectural state**
- Run sequence
- Collect **architectural state**



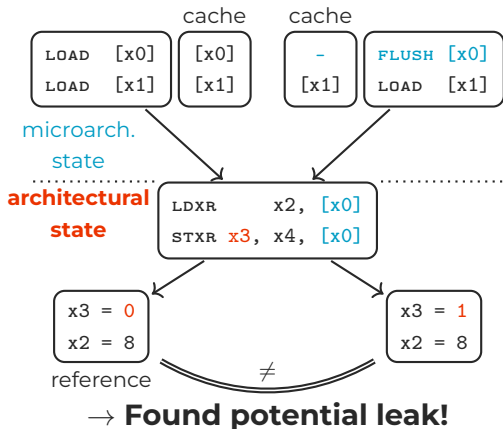
ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime **microarchitectural state**
- Run sequence
- Collect **architectural state**
- Compare to reference state



ExfilState: Finding a Leak



- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state
- Run sequence
- Collect architectural state
- Compare to reference state



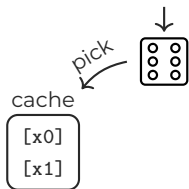
ExfilState: Identifying a Good Leak



- Repeat 5k times:



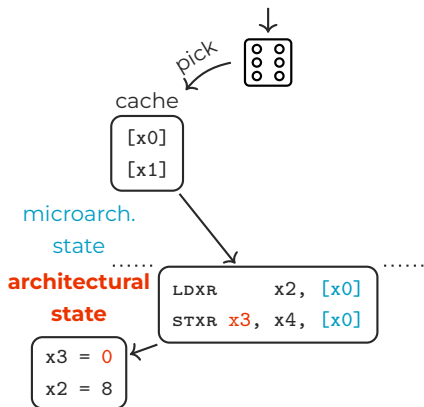
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state



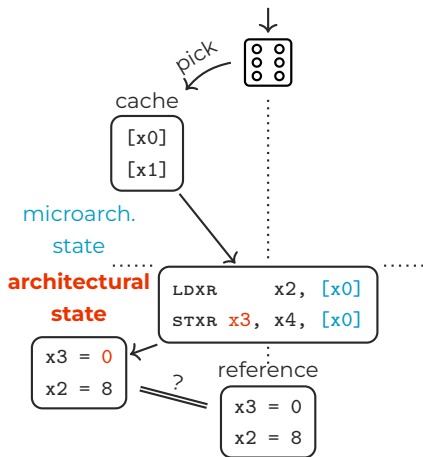
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**



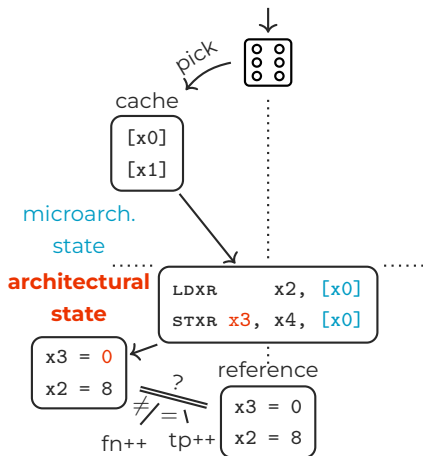
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**
 - Compare to reference state



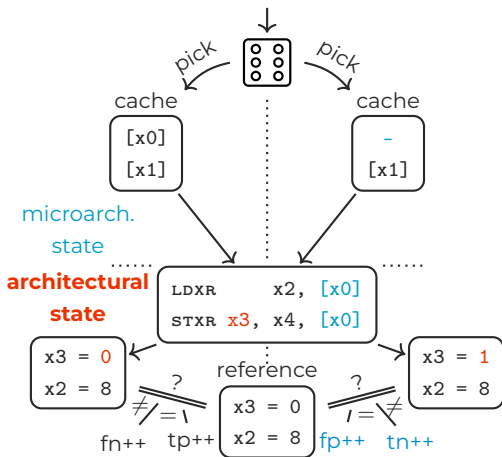
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**
 - Compare to reference state



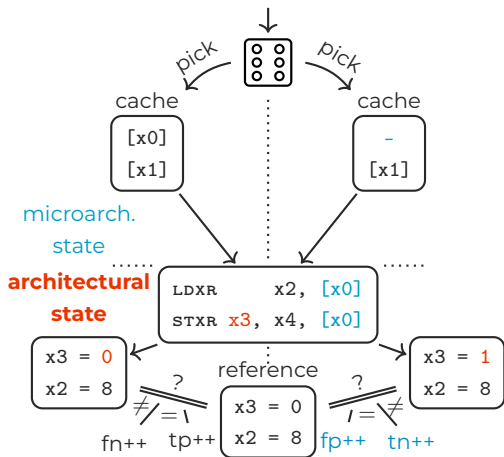
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**
 - Compare to reference state



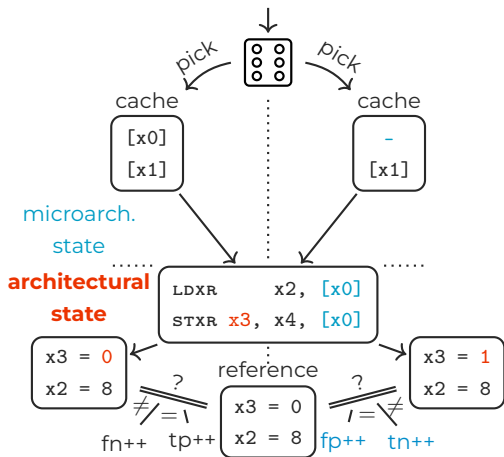
ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**
 - Compare to reference state
- Compute F-score



ExfilState: Identifying a Good Leak



- Repeat 5k times:
 - Randomly pick discovered cache state
 - Prime **microarchitectural state**
 - Run sequence
 - Collect **architectural state**
 - Compare to reference state
- Compute F-score
- If high: **Found Leaking Sequence!**

Manual review has a combinatorics problem



There are more instruction sequences than there are coffee breaks



ExfilState: Fuzzing Results

160 devices

37 ARM microarchitectures





ExfilState: Fuzzing Results

160 devices

37 ARM microarchitectures



Side Channel	Cortex-A53	Cortex-A55	Cortex-A510	Cortex-A520	Cortex-A72	Cortex-A73	Cortex-A75	Cortex-A76	Cortex-A77	Cortex-A78	Cortex-A710	Cortex-A715	Cortex-A720	Cortex-A725	Cortex-X1	Cortex-X2	Cortex-X3	Cortex-X4	Kryo	Falkor-V1/Kryo	Kryo-V2	Kryo-3XX-Gold	Kryo-3XX-Silver	Kryo-4XX-Gold	Kryo-4XX-Silver	Carmel	Kunpeng Pro	Oryon	Oryon V2 P-L	Oryon V2 P-M	Exynos M3	Neoverse-N1	Neoverse-V2	Firestorm-M1	Icestorm-M1	Avalanche-M2	Blizzard-M2
LX+Sx	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓						✓	✓	✓	~		✓		✓	✓	✓	✓			✓	✓	✓	✓	~
STORE+RET	✓	✓	✓	✓		~		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	~	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SPLIT-STORE						✓	✓												✓		✓																
TRANSLATION-RACE						✓	✓												✓		✓																
POINTER-CHASE	✓				✓																																
Affected	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓



ExfilState: Fuzzing Results

160 devices

37 ARM microarchitectures



Side Channel	Cortex-A53	Cortex-A55	Cortex-A510	Cortex-A520	Cortex-A72	Cortex-A73	Cortex-A75	Cortex-A76	Cortex-A77	Cortex-A78	Cortex-A710	Cortex-A715	Cortex-A720	Cortex-A725	Cortex-X1	Cortex-X2	Cortex-X3	Cortex-X4	Kryo	Falkor-V1/Kryo	Kryo-V2	Kryo-3XX-Gold	Kryo-3XX-Silver	Kryo-4XX-Gold	Kryo-4XX-Silver	Carmel	Kunpeng Pro	Oryon	Oryon V2 P-L	Oryon V2 P-M	Exynos M3	Neoverse-N1	Neoverse-V2	Firestorm-M1	Icestorm-M1	Avalanche-M2	Blizzard-M2	
Lx+Sx	✓	✓	✓	✓	✓	✓	✓	✓				✓	✓	✓						✓	✓	✓	~		✓		✓	✓	✓	✓			✓	✓	✓	✓	✓	~
STORE+RET	✓	✓	✓	✓		~		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	~	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SPLIT-STORE						✓	✓													✓		✓																
TRANSLATION-RACE						✓	✓													✓		✓																
POINTER-CHASE	✓				✓																																	
Affected	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓



How do such sequences look like? — Lx+Sx

```
LDXR    x2, [victim]  
STXR    x3, x4, [victim]
```



How do such sequences look like? — Lx+Sx

```
LDXR    x2, [victim]  
STXR    x3, x4, [victim]
```

- Load **victim**
- Mark as **exclusive access**



How do such sequences look like? — Lx+Sx

```
LDXR    x2, [victim]  
STXR    x3, x4, [victim]
```

- Load **victim**
- Mark as **exclusive access**
- Store to **victim**
- Check **exclusive access**
- **x3**=0: Success
- **x3**=1: Failure



What do such sequences look like? — Lx+Sx

```
LDXR    x2, [victim]  
STXR    x3, x4, [victim]
```

- ExfilState uncovers:
victim in cache $\iff$ **x3**=0



What do such sequences look like? — Lx+Sx

```
LDXR    x2, [victim]  
STXR    x3, x4, [victim]
```

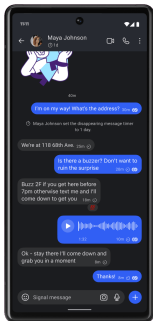
- ExfilState uncovers:
 victim in cache $\iff$ x3=0
- x3 architecturally leaks cache state

“Why is this a security problem?”





Leaking User Input on Android



shared library
libinput.so





Leaking User Input on Android



shared library
libinput.so





Leaking User Input on Android



shared library
libinput.so



interrupt





Leaking User Input on Android



shared library
libinput.so

←
schedule





Leaking User Input on Android



query

shared library
libinput.so





Leaking User Input on Android



query

shared library
libinput.so

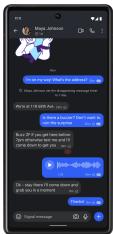




Leaking User Input on Android

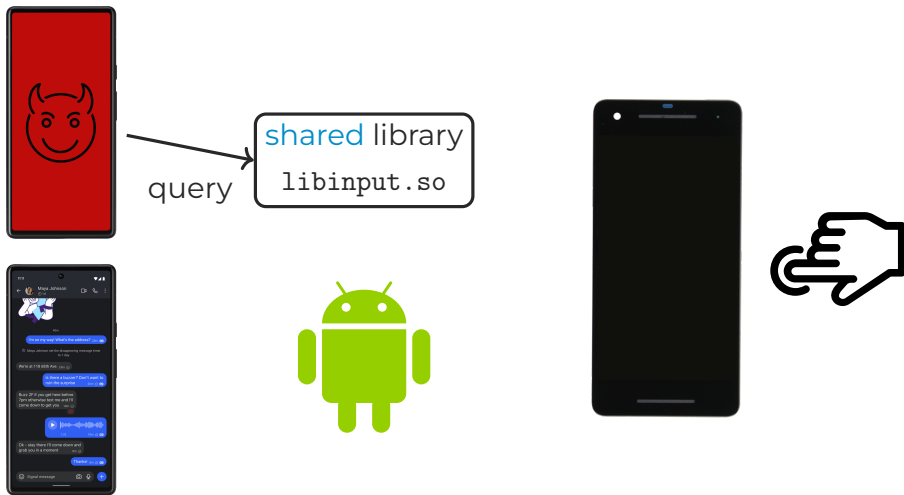


shared library
libinput.so



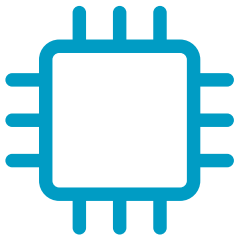
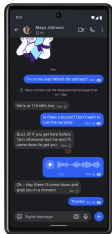


Leaking User Input on Android



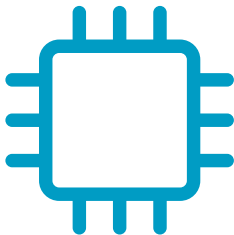
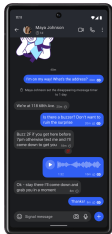


Leaking User Input on Android



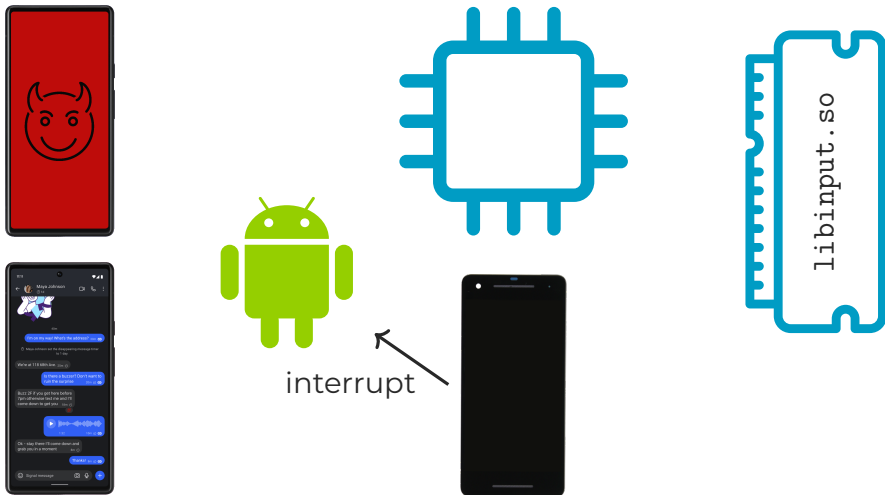


Leaking User Input on Android



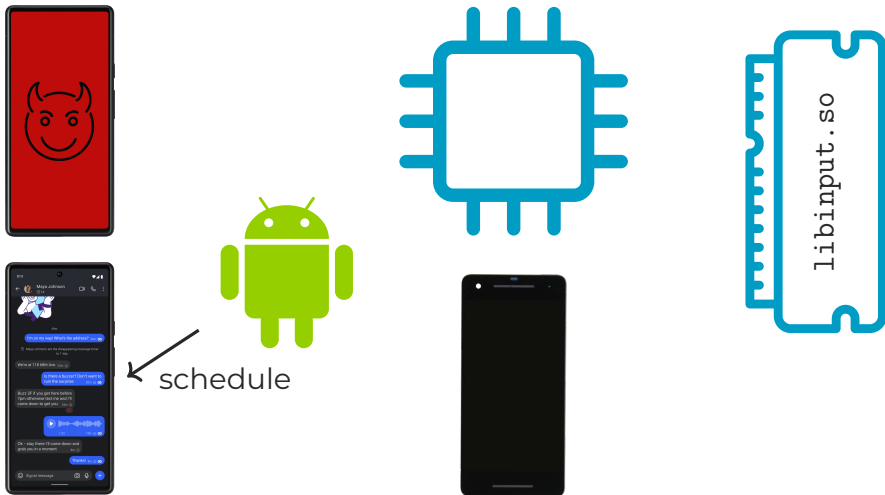


Leaking User Input on Android



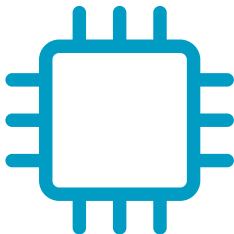
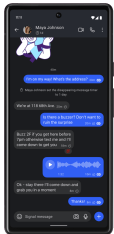


Leaking User Input on Android





Leaking User Input on Android

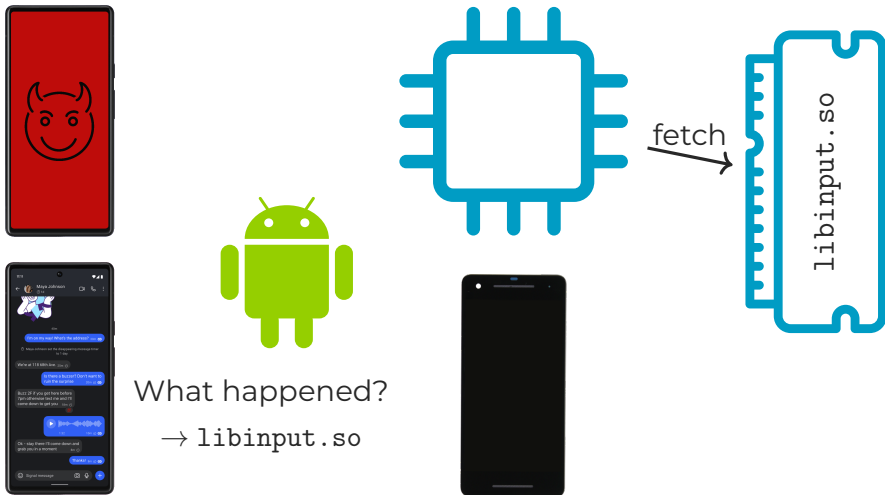


What happened?
→ `libinput.so`



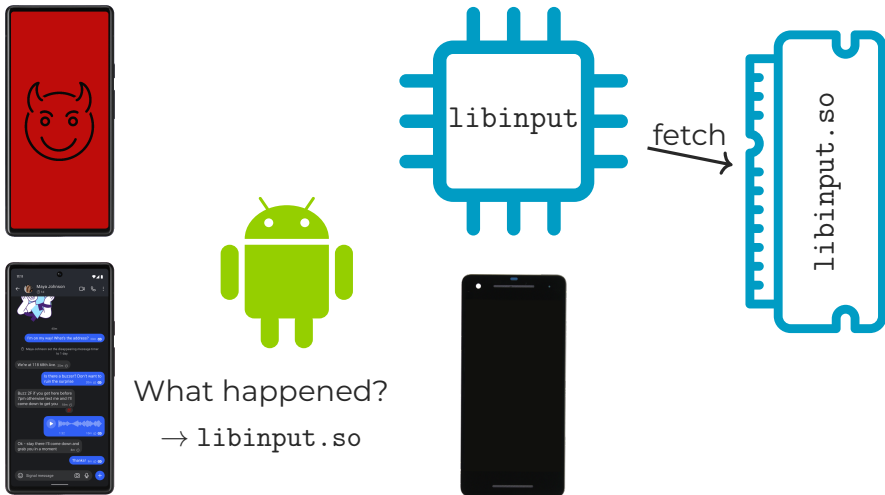


Leaking User Input on Android



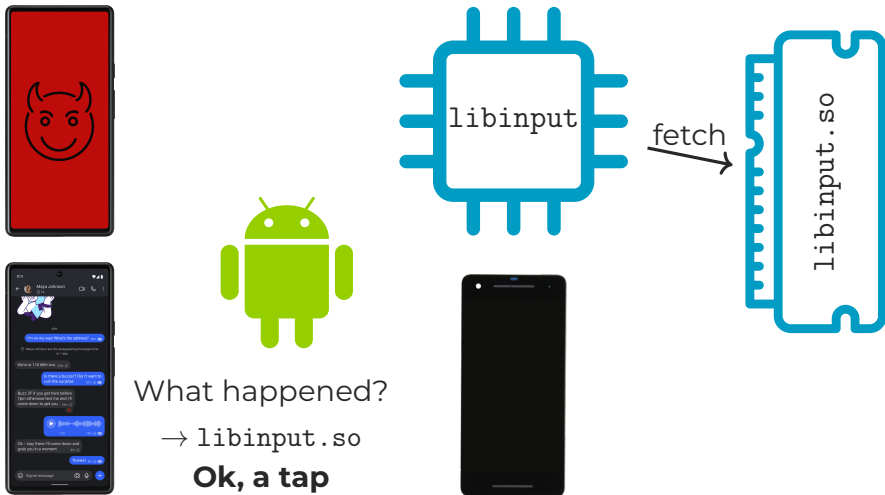


Leaking User Input on Android



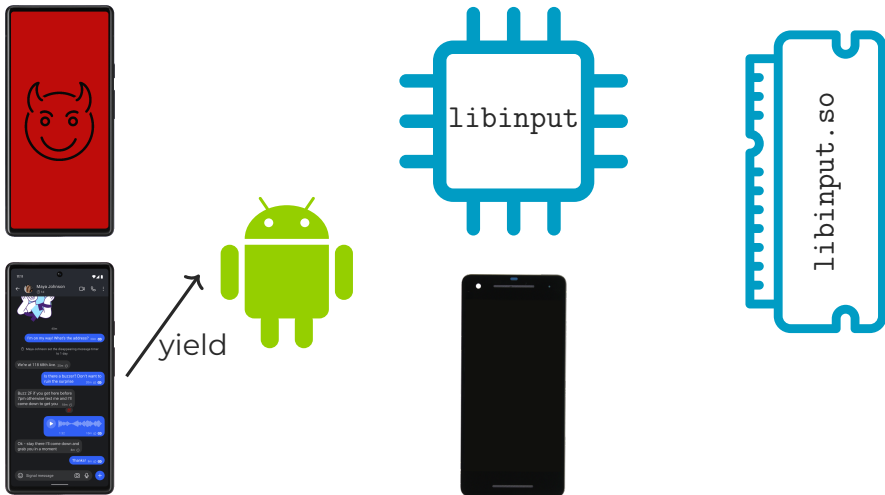


Leaking User Input on Android



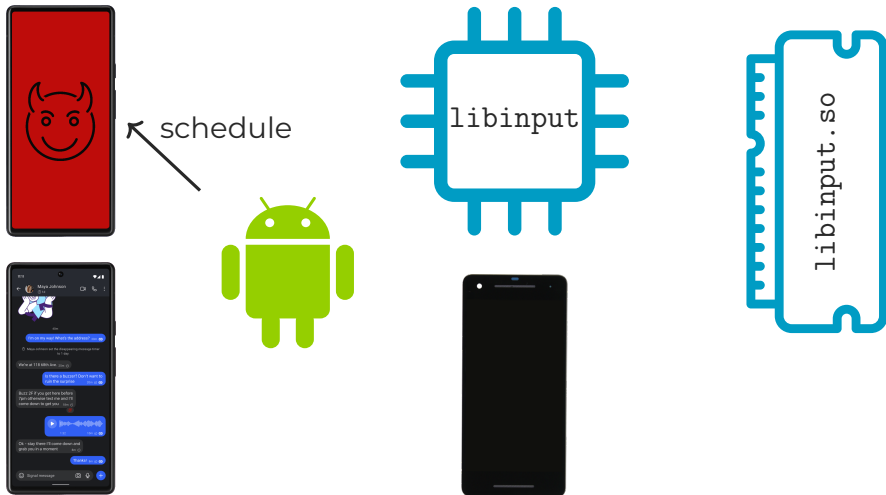


Leaking User Input on Android



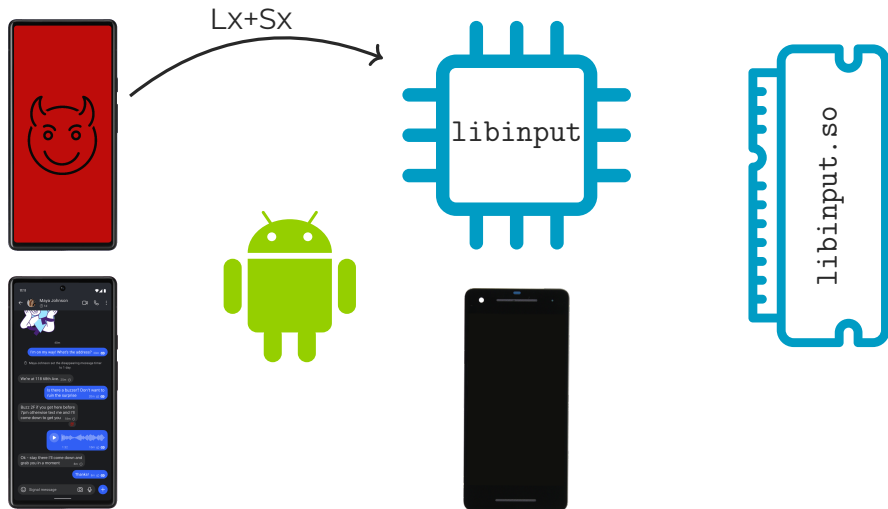


Leaking User Input on Android



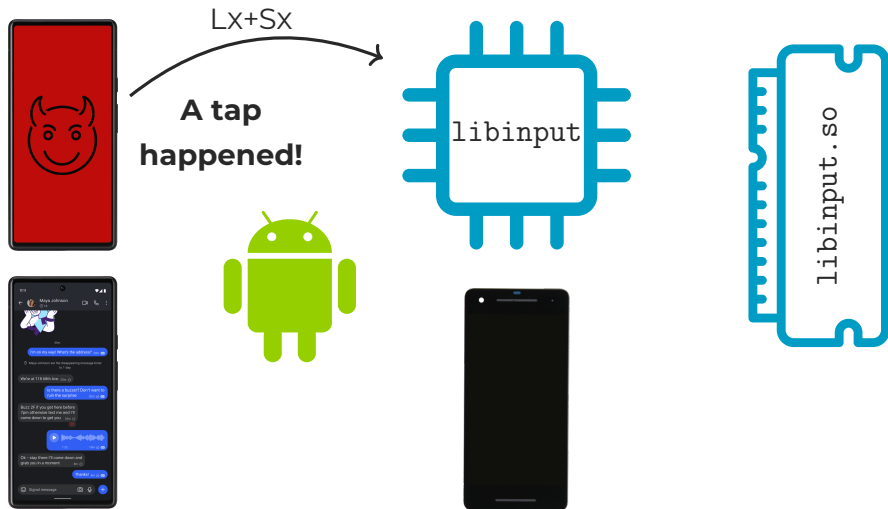


Leaking User Input on Android



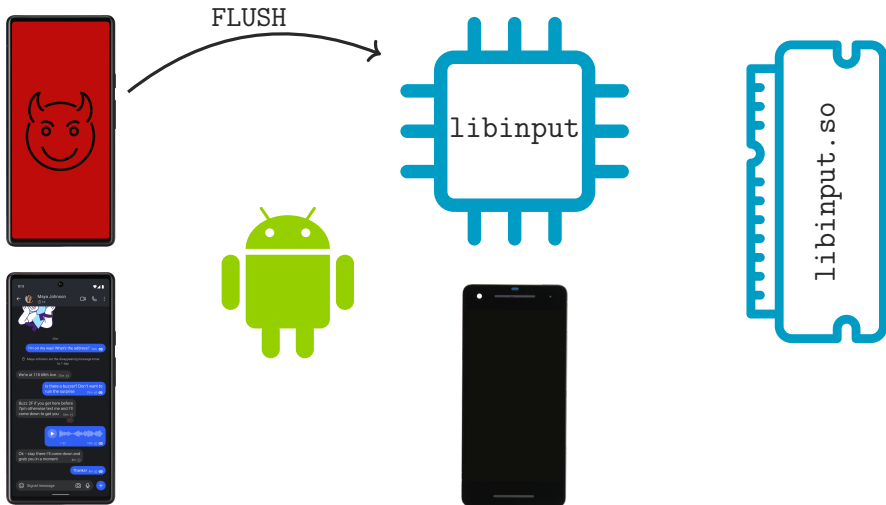


Leaking User Input on Android



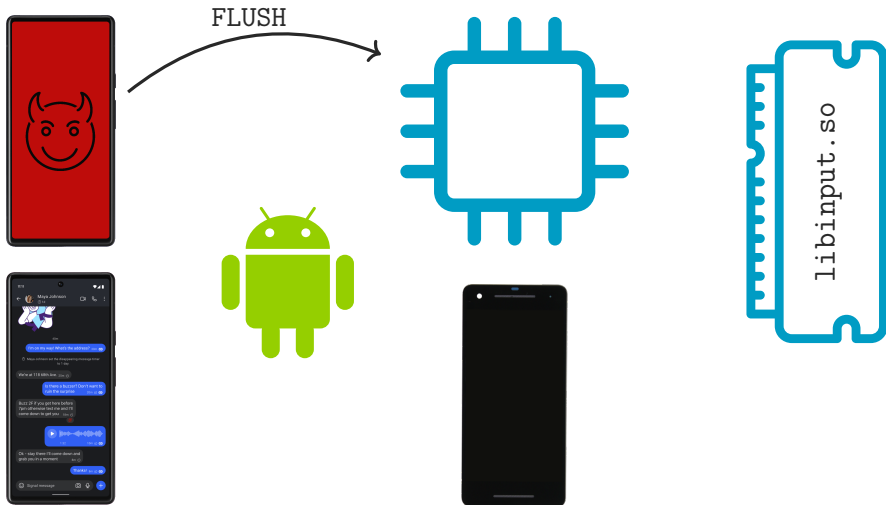


Leaking User Input on Android



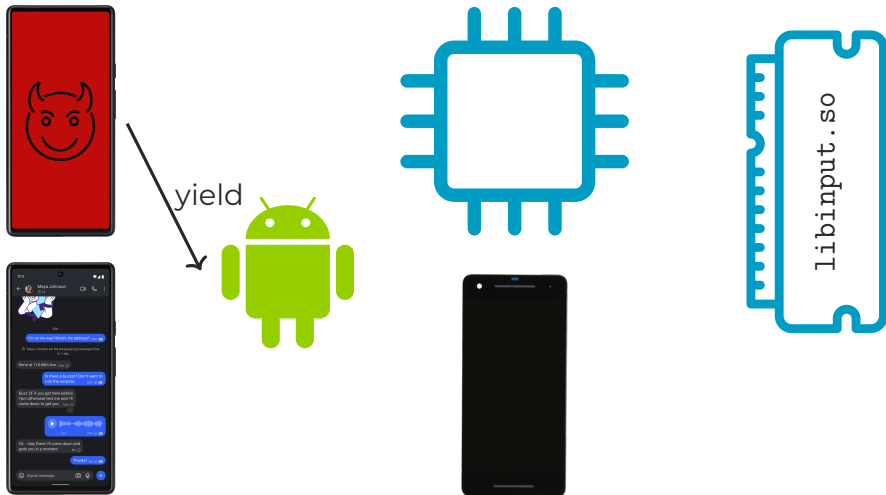


Leaking User Input on Android



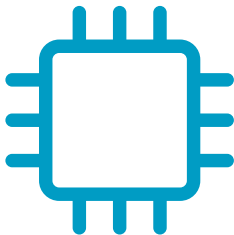
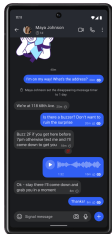


Leaking User Input on Android



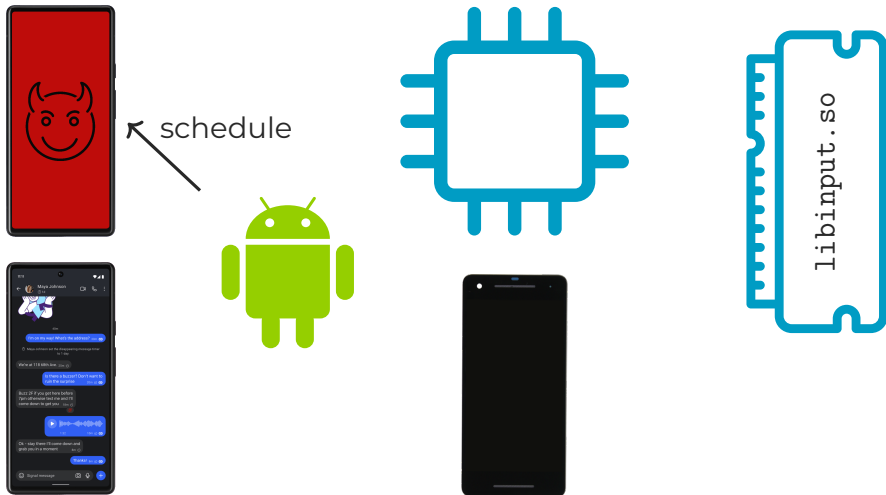


Leaking User Input on Android



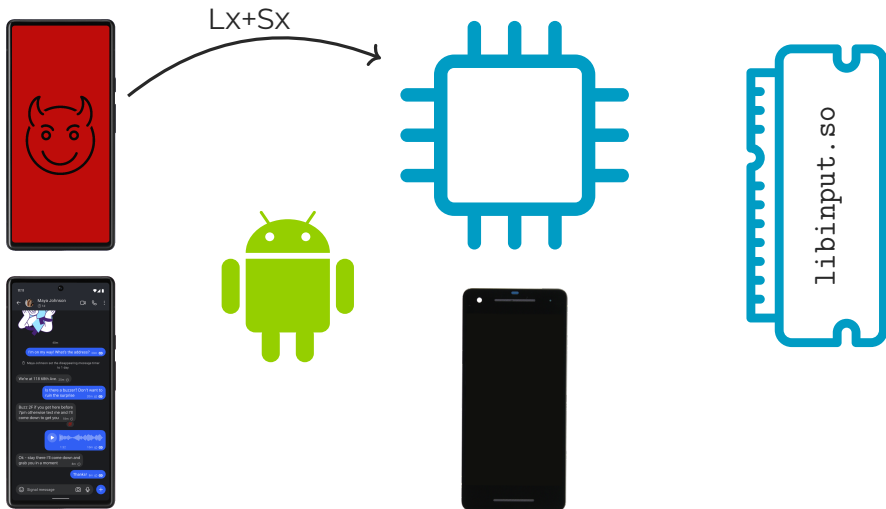


Leaking User Input on Android



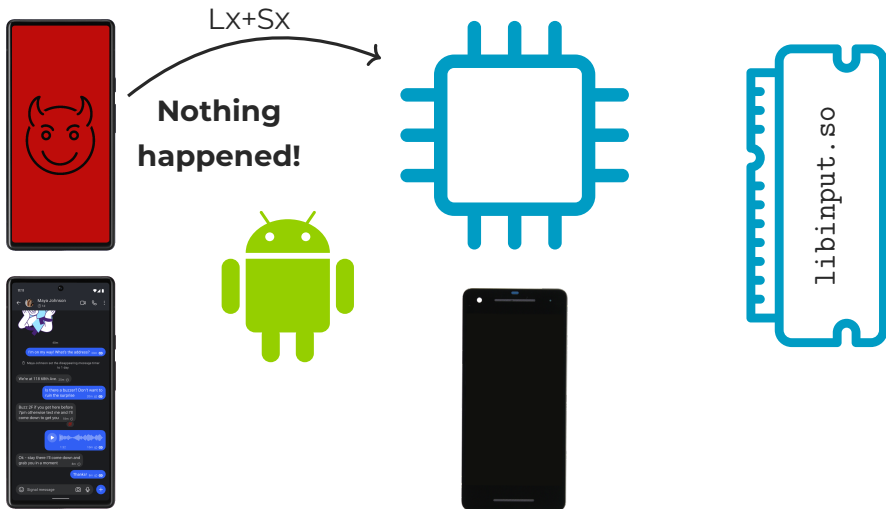


Leaking User Input on Android

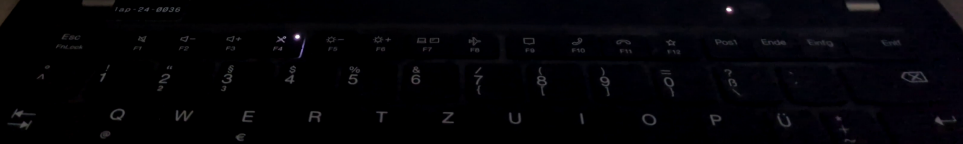




Leaking User Input on Android



- Tap
- Tap
- Tap
- Tap
- Tap
- Tap
- Tap





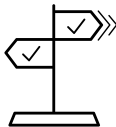
Takeaway



Timers are optional
for side channels



Architectural
boundary gets blurry



Differential testing
finds leaks

Takeaway

Tomorrow we'll ask the harder question:

Can the CPU leak the data directly?