CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

No Time To Leak

Exposing Timer-Free Cache-State Leaks on ARM CPU

Fabian Thomas | PhD Student @ CISPA

Flush+Reload

Flush+Reload

Flush+Reload

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

Flush+Reload

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

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=180910
<rdar://problem/36085943>

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

rniwa committed on Jan 8, 2018

```
Reduce the precision of "high" resolution time to 1ms
https://bugs.webkit.org/show_bug.cgi?id=180910
<rdar://problem/36085943>

Reviewed by Saam Barati.

Source/WebCore:

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

What if we have no timer at all?

Flush+Reload

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

Flush+Reload

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

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

ExfilState: Finding a Leak

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

- Generate random instruction sequence

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

cache

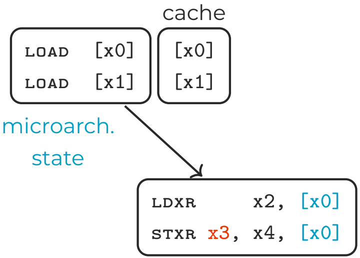
```
LOAD  [x0]    [x0]
LOAD  [x1]    [x1]
```

microarch.
state

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

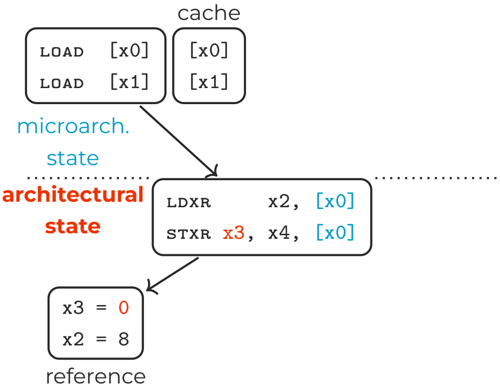
- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state

ExfilState: Finding a Leak



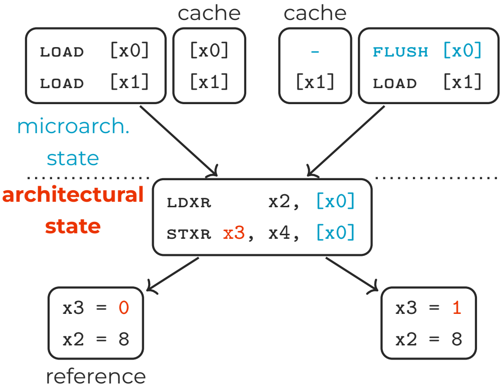
- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state
- Run sequence

EXFILSTATE: Finding a Leak



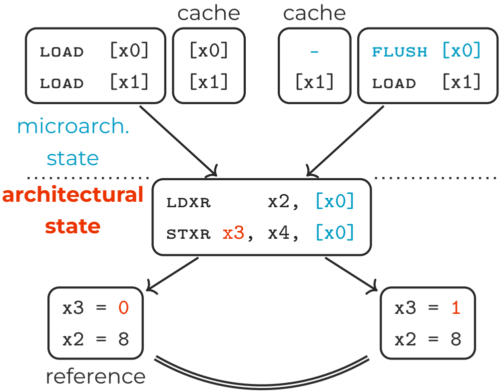
- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state
- Run sequence
- Collect architectural state

ExfilState: Finding a Leak



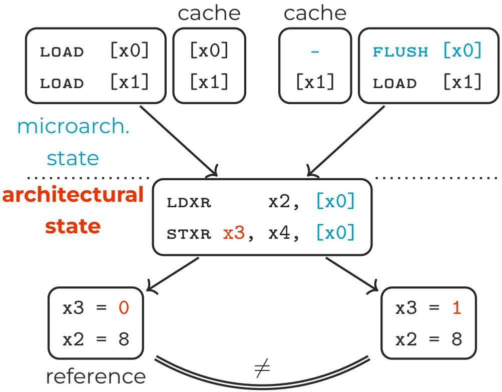
- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state
- Run sequence
- Collect architectural state

EXFILSTATE: Finding a Leak



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

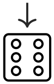
EXFILSTATE: Finding a Leak



→ **Found potential leak!**

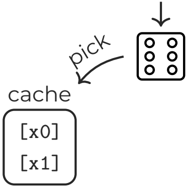
- Generate random instruction sequence
- Generate cache state
- Prime microarchitectural state
- Run sequence
- Collect architectural state
- Compare to reference state

ExfilState: Identifying a Good Leak



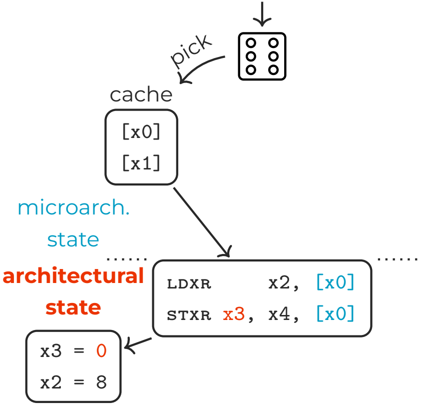
- Repeat 5k times:

EXFILSTATE: Identifying a Good Leak



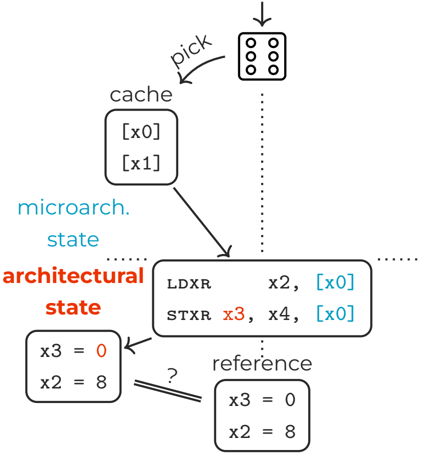
- Repeat 5k times:
  - Randomly pick discovered cache state

EXFILSTATE: Identifying a Good Leak



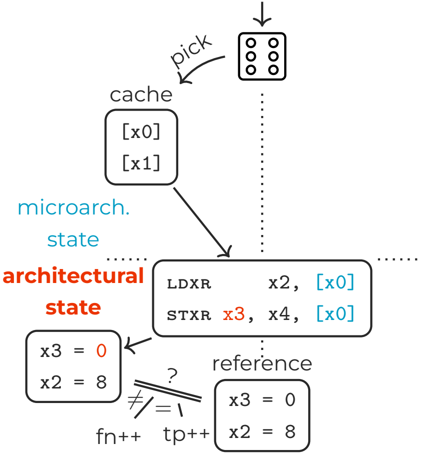
- Repeat 5k times:
  - Randomly pick discovered cache state
  - Prime microarchitectural state
  - Run sequence
  - Collect architectural state

EXFILSTATE: Identifying a Good Leak



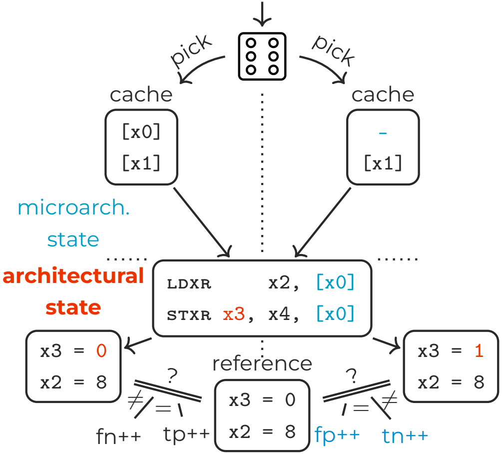
- Repeat 5k times:
  - Randomly pick discovered cache state
  - Prime microarchitectural state
  - Run sequence
  - Collect architectural state
  - Compare to reference state

EXFILSTATE: Identifying a Good Leak



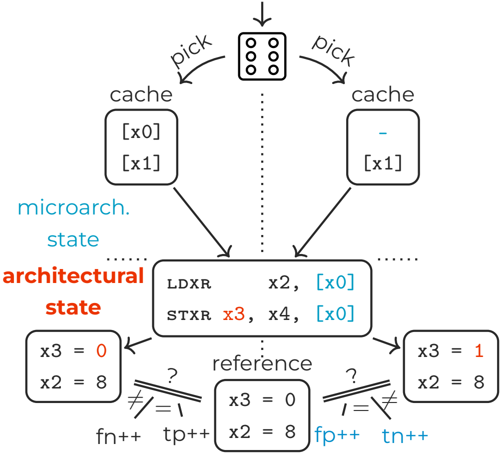
- Repeat 5k times:
  - Randomly pick discovered cache state
  - Prime microarchitectural state
  - Run sequence
  - Collect architectural state
  - Compare to reference state

EXFILSTATE: Identifying a Good Leak



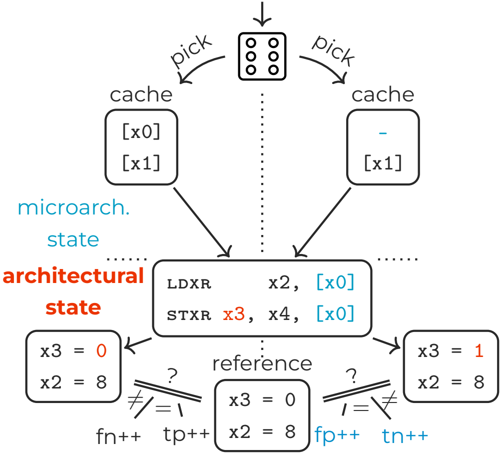
- Repeat 5k times:
  - Randomly pick discovered cache state
  - Prime microarchitectural state
  - Run sequence
  - Collect architectural state
  - Compare to reference state

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!**

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
Lx+Sx	$\checkmark^{2}$	$\checkmark^{4}$	$\checkmark^{4}$	$\checkmark^{5}$	$\checkmark^{2}$	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$				$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$				
Store+Ret	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$		$\sim^{2}$		$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$
Split-Store						$\checkmark^{2}$	$\checkmark^{2}$											
Translation-Race						$\checkmark^{2}$	$\checkmark^{2}$											
Pointer-Chase	$\checkmark^{3}$				$\checkmark^{2}$													
Affected	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Side Channel	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		$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{4}$	$\sim^{4}$		$\checkmark^{4}$			$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{4}$			$\checkmark^{4}$	$\checkmark^{5}$	$\checkmark^{4}$	$\sim^{8}$
Store+Ret			$\checkmark^{2}$	$\sim^{2}$	$\checkmark^{3}$	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{2}$	$\checkmark^{4}$	$\checkmark^{2}$	$\checkmark^{3}$	$\checkmark^{3}$	$\checkmark^{2}$	$\checkmark^{2}$	$\checkmark^{2}$	$\checkmark^{4}$	$\checkmark^{2}$	$\checkmark^{3}$
Split-Store		$\checkmark^{2}$		$\checkmark^{2}$															
Translation-Race		$\checkmark^{2}$		$\checkmark^{2}$															
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	✓^{2}	✓^{4}	✓^{4}	✓^{5}	✓^{2}	✓^{2}	✓^{3}	✓^{3}				✓^{3}	✓^{3}	✓^{3}						✓^{2}	✓^{3}	✓^{4}	∼^{4}		✓^{4}			✓^{3}	✓^{3}	✓^{3}	✓^{4}			✓^{4}	✓^{5}	✓^{4}	∼^{8}
Store+Ret	✓^{2}	✓^{3}	✓^{3}	✓^{3}		∼^{2}		✓^{2}	✓^{3}	✓^{2}	✓^{3}	✓^{3}	✓^{3}	✓^{2}	✓^{3}	✓^{3}	✓^{3}	✓^{3}			✓^{2}	∼^{2}	✓^{3}	✓^{2}	✓^{3}	✓^{3}	✓^{2}	✓^{4}	✓^{2}	✓^{3}	✓^{3}	✓^{2}	✓^{2}	✓^{2}	✓^{4}	✓^{2}	✓^{3}
Split-Store						✓^{2}	✓^{2}													✓^{2}		✓^{2}															
Translation-Race						✓^{2}	✓^{2}													✓^{2}		✓^{2}															
Pointer-Chase	✓^{3}				✓^{2}																																
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

How do such sequences look like? — Lx+Sx

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

- ExfilState uncovers:
  `victim` in cache $\Longleftrightarrow$ `x3`=0

How do such sequences look like? — Lx+Sx

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

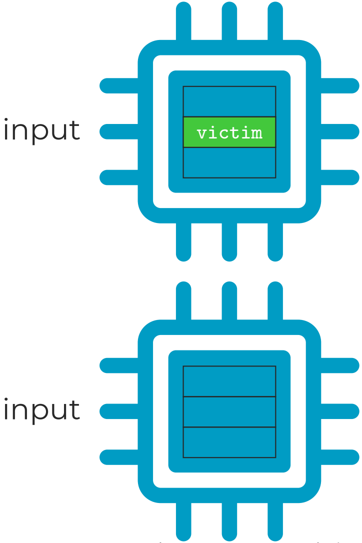
- ExfilState uncovers:
  `victim` in cache $\Longleftrightarrow$ `x3`=0
- `x3` architecturally leaks cache state

How do such sequences look like? — Lx+Sx

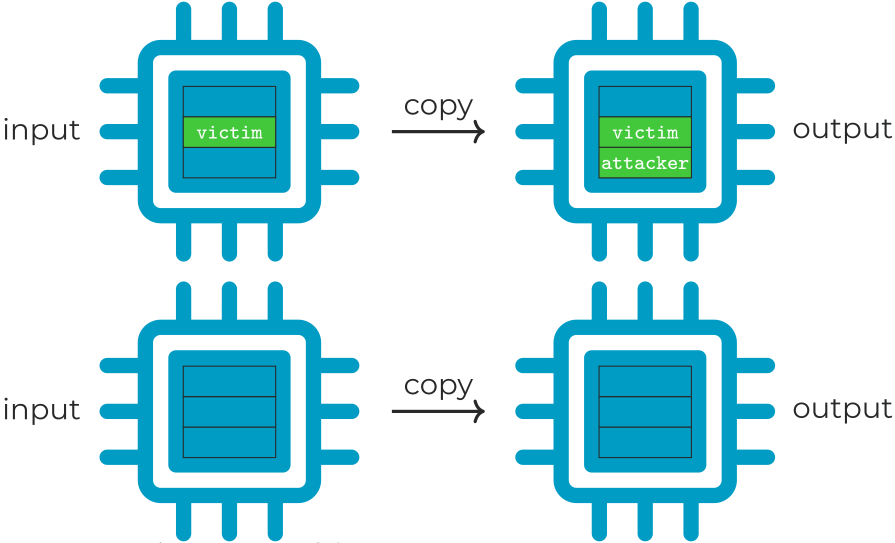
```
LDXR      x2, [victim]
STXR x3, x4, [victim]
```

- ExfilState uncovers:
  `victim` in cache $\iff$ `x3`=0
- `x3` architecturally leaks cache state
- **Problem:** victim needs to be writable

How to exploit? — Cache-State Copier



How to exploit? — Cache-State Copier



How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
  BRANCH x1
LOAD x2, VICTIM

LOAD x4, [ATTACKER+x2]
```

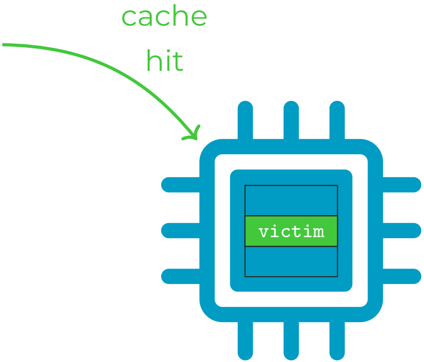
Triggers transient execution

How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM

LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

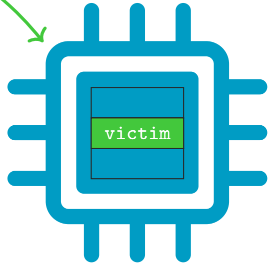


How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM
LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

cache
hit



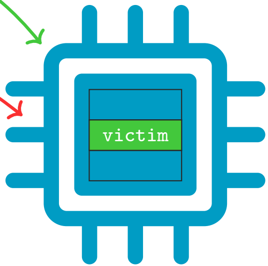
How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM
LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

cache hit

cache miss



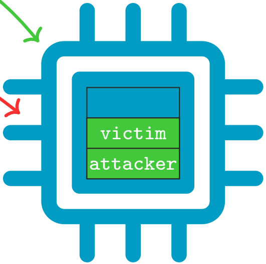
How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM
LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

cache hit

cache miss



How to exploit? — Cache-State Copier

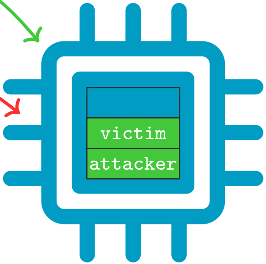
```
LOAD x1, [x2]
BRANCH x1
```

Triggers transient execution

```
LOAD x2, VICTIM
LOAD x4, [ATTACKER+x2]
```

cache hit

cache miss

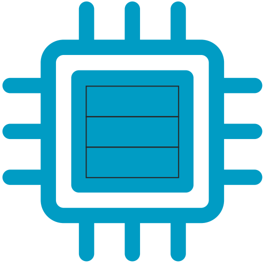


How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
  BRANCH x1
LOAD x2, VICTIM

LOAD x4, [ATTACKER+x2]
```

Triggers transient execution



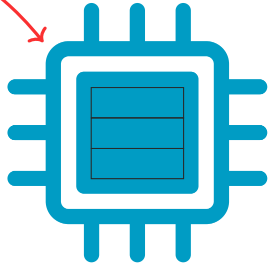
How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM

LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

cache miss

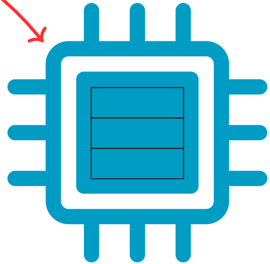


How to exploit? — Cache-State Copier

```
LOAD x1, [x2]
BRANCH x1
LOAD x2, VICTIM
LOAD x4, [ATTACKER+x2]
```

Triggers transient execution

cache miss

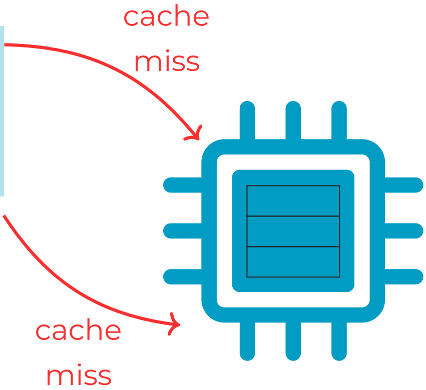


How to exploit? — Cache-State Copier

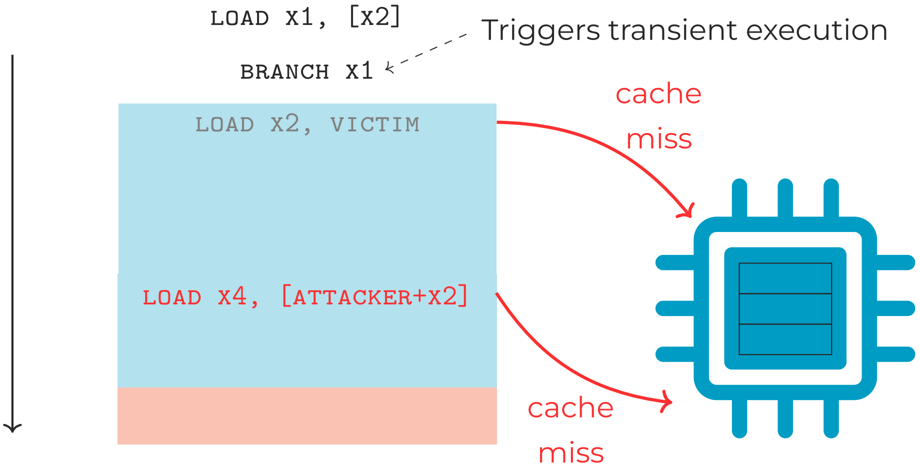
```
LOAD x1, [x2]
  BRANCH x1
LOAD x2, VICTIM

LOAD x4, [ATTACKER+x2]
```

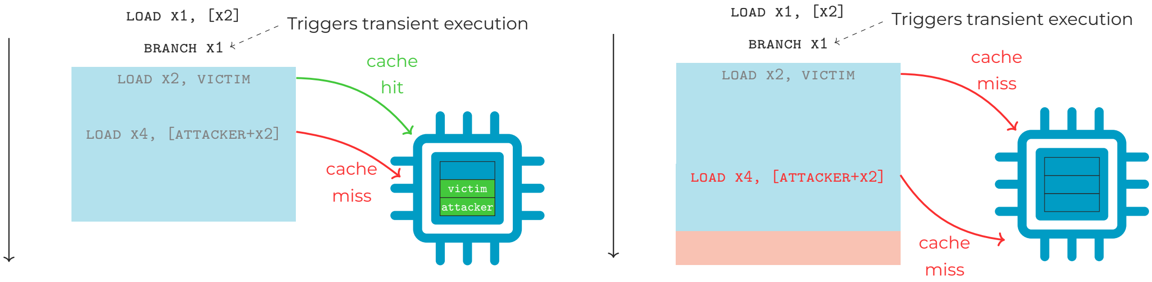
Triggers transient execution



How to exploit? — Cache-State Copier



How to exploit? — Cache-State Copier



attacker in cache $\iff$ victim in cache

$\rightarrow$ mount Lx+Sx

Summary

- How to mount side channel attacks in a world without timers?

fabianthomas.de

github.com/cispa/exfilstate

Summary

Fabian Thomas, Michael Torres, Daniel Moghimi, Michael Schwarz @ CCS '25

- How to mount side channel attacks in a world without timers?
- ExfilState automatically discovers **timerless cache-state leakage oracles**

fabianthomas.de

github.com/cispa/exfilstate

Summary

- How to mount side channel attacks in a world without timers?
- ExfilState automatically discovers **timerless cache-state leakage oracles**
- Uses **correlation-analysis** for ranking

fabianthomas.de

github.com/cispa/exfilstate

Summary

- How to mount side channel attacks in a world without timers?
- ExfilState automatically discovers **timerless cache-state leakage oracles**
- Uses **correlation-analysis** for ranking
- Discovered **5 side channels** on **37 ARM microarchitectures**

fabianthomas.de

github.com/cispa/exfilstate

Summary

- How to mount side channel attacks in a world without timers?
- ExfilState automatically discovers **timerless cache-state leakage oracles**
- Uses **correlation-analysis** for ranking
- Discovered **5 side channels** on **37 ARM microarchitectures**
- Exploitation requires **cache-state copying**

fabianthomas.de

github.com/cispa/exfilstate

AES T-table

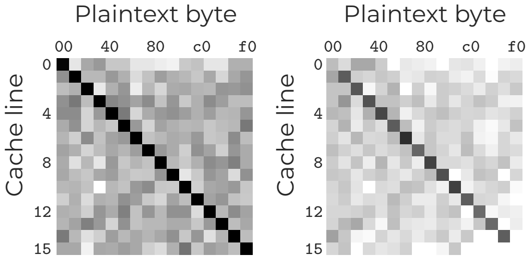


Figure 1: AES T-table cache-access pattern on an ARM64 Cortex-A72 with Lx+Sx (left) and Flush+Reload (right).

Lx+Sx

```
1    STTRH       w1, [x2]
2    LDAXRH      w2, [x1]
3    LDRSB       x0, [x3]
4    STLXRB  x0, w3, [x1]
```

Figure 2: Lx+Sx on Apple Avalanche. `x1` is the victim cache line. `x2` and `x3` are unrelated cache lines. Instructions 1 and 3 are not needed on Cortex-A73.

Manufacturer	Models
Apple	Mac Mini (M1), Mac Mini (M2)
ASUS	ZenFone Max M1, ZenFone Max M2
AWS	EC2 M8g Graviton4
FriendlyELEC	NanoPi R6S
Fujitsu	Arrows Be3
Globalscale	MOCHAbin
Google	Pixel Watch, Pixel Watch 2, Pixel, Pixel 2, Pixel 2 XL, Pixel 3, Pixel 4a, Pixel 5, Pixel 5a, Pixel 6, Pixel 6 Pro, Pixel 6a, Pixel 7, Pixel 7 Pro, Pixel 7a, Pixel 8, Pixel 8 Pro, Pixel 8a, Pixel 9, Pixel 9 Pro, Pixel 9 Pro Fold, Pixel 9 Pro XL, Pixel Fold, Pixel Tablet
Google Cloud	C4A Axion, T2A Ampere
Hardkernel	ODROID-N2+, Odroid-C4
HUAWEI	P20 Lite, Mate 9
Lenovo	Tab P11, Tab P12
LG	Velvet
Motorola	Moto G31, Moto G54 5G, Moto G30, Moto G 5G (2022), Moto G Play (2024), Razr 5G, Razr+ (2024), Moto Z Force
Nokia	C31
Nothing	Phone (1)
NVIDIA	Jetson Orin Nano
OnePlus	10T, 11, Nord CE 3 Lite, 7 Pro (US Version), 6T, Nord 2T, Nord 3, Ace 2V, 9 Pro
OPPO	A53, A79 5G, Find X3 Lite, Reno10 Pro+ 5G
OrangePi	Kunpeng Pro
Poco	X6 Pro, M6 Pro 5G, F3
Qualcomm	Dell XPS 13
Radxa	ROCK 3A, ROCK 5C
Raspberry Pi	4 Model B, 5
Realme	GT Neo 3T, GT Master Edition
Samsung	Galaxy S7 edge (Verizon), Galaxy Note 5 (Verizon), Galaxy Tab S3 (Verizon), Galaxy A02s, Galaxy A8, Galaxy A10, Galaxy A12, Galaxy A14 5G, Galaxy A14 5G, Galaxy A15, Galaxy A25 5G, Galaxy A35 5G, Galaxy A51, Galaxy A52s 5G, Galaxy A54 5G, Galaxy Z Flip 3 5G, Galaxy Z Flip 4, Galaxy Z Flip 5, Galaxy Z Fold 2 5G, Galaxy Z Fold 4, Galaxy Z Fold 5, Galaxy J7 Prime, Galaxy A51 5G, Galaxy S8, Galaxy S8+, Galaxy S9, Galaxy S9, Galaxy S9+, Galaxy S20 5G, Galaxy S21 5G, Galaxy S21+ 5G, Galaxy S21 Ultra 5G, Galaxy M11, Galaxy Note 9, Galaxy Note 9, Galaxy S23 FE, Galaxy S22 5G, Galaxy S22+ 5G, Galaxy S22 Ultra 5G, Galaxy S23+, Galaxy S23 Ultra, Galaxy S23 Ultra, Galaxy S24, Galaxy S24 Ultra, Galaxy S24 Ultra, Galaxy Tab A7 Lite LTE, Galaxy Tab A 10.1 (2016) Wi-Fi, Galaxy Tab S7 FE 5G, Galaxy Tab A8 Wi-Fi, Galaxy Tab A9 Wi-Fi, Galaxy Tab S9 FE+, Galaxy Tab S7 Wi-Fi, Galaxy Tab S8 Ultra Wi-Fi
Sharp	AQUOS sense2
Sony	Xperia XZ1 Compact, Xperia 10 V, Xperia 1 V, Xperia XZ, Xperia 10 II
Vivo	Y20s [G], Y31 (2021), Y73, X60, Y02s, Y95, Y12, Y15, U3x, Y17
Xiaomi	Mi 11i, Mi A2 Lite, Redmi Note 13 Pro+ 5G, Redmi Note 11 5G, Redmi Note 11 Pro, Redmi Note 12, Redmi Note 12 Pro+
Zebra	TC77

Microarchitecture	Lx+Sx			Store+Ret			Split-Store			Translation-Race			Pointer-Chase			Affected
Cortex-A53	0.00	100.00	✓[2]	0.00	100.00	✓[2]							0.00	99.42	✓[3]	✓
Cortex-A55	0.00	99.86	✓[4]	0.00	100.00	✓[3]										✓
Cortex-A510	19.63	86.31	✓[4]	1.27	98.27	✓[3]										✓
Cortex-A520	16.90	80.59	✓[5]	1.47	96.85	✓[3]										✓
Cortex-A72	0.00	100.00	✓[2]										0.00	99.94	✓[2]	✓
Cortex-A73	0.00	100.00	✓[2]	27.45	88.44	∼[2]	0.00	99.70	✓[2]	0.40	98.71	✓[2]				✓
Cortex-A75	0.00	99.05	✓[3]				0.00	99.57	✓[2]	0.00	100.00	✓[2]				✓
Cortex-A76	7.04	92.90	✓[3]	0.00	99.92	✓[2]										✓
Cortex-A77				0.00	99.81	✓[3]										✓
Cortex-A78				0.00	99.77	✓[2]										✓
Cortex-A710				0.00	100.00	✓[3]										✓
Cortex-A715	0.00	99.90	✓[3]	0.00	100.00	✓[3]										✓
Cortex-A720	0.00	94.46	✓[3]	0.00	99.74	✓[3]										✓
Cortex-A725	0.00	88.43	✓[3]	0.00	99.74	✓[2]										✓
Cortex-X1				0.00	99.89	✓[3]										✓
Cortex-X2				0.00	99.94	✓[3]										✓
Cortex-X3				0.20	99.44	✓[3]										✓
Cortex-X4				0.00	99.98	✓[3]										✓
Kryo Falkor-V1/Kryo	0.00	99.85	✓[2]				0.00	99.61	✓[2]	0.20	98.21	✓[2]				✓
Kryo-V2	0.20	99.83	✓[3]	0.00	99.36	✓[2]										✓
Kryo-3XX-Gold	0.10	99.00	✓[4]	35.45	98.43	∼[2]	0.00	99.50	✓[2]	0.00	99.56	✓[2]				✓
Kryo-3XX-Silver	30.57	95.93	∼[4]	1.27	95.55	✓[3]										✓
Kryo-4XX-Gold				0.00	99.93	✓[2]										✓
Kryo-4XX-Silver	1.96	98.92	✓[4]	0.98	98.41	✓[3]										✓
Carmel				4.59	97.46	✓[3]										✓
Kunpeng Pro				0.00	100.00	✓[2]										✓
Oryon	0.10	99.26	✓[3]	0.10	96.19	✓[4]										✓
Oryon V2 Phoenix L	0.49	83.42	✓[3]	0.00	98.81	✓[2]										✓
Oryon V2 Phoenix M	0.00	81.40	✓[3]	0.00	99.00	✓[3]										✓
Exynos M3	1.57	97.54	✓[4]	2.06	99.77	✓[3]										✓
Neoverse-N1				0.00	99.89	✓[2]										✓
Neoverse-V2				0.00	99.88	✓[2]										✓
Firestorm-M1	0.00	98.52	✓[4]	0.00	80.47	✓[2]										✓
Icestorm-M1	7.43	90.80	✓[5]	0.00	99.88	✓[4]										✓
Avalanche-M2	5.47	90.15	✓[5]	0.59	84.46	✓[2]										✓
Blizzard-M2	45.32	80.79	∼[8]	5.28	93.27	✓[3]										✓

EXFILSTATE: 5 Side Channels Discovered

```
LDXR x1, [victim]
STXR w0, x2, [victim]
```

Figure 3: LX+SX: `STXR` (store exclusive) succeeds (status in `w0`) only if victim cached.

```
; w0 enc 'MOV x3,#0', w1 'MOV x3,#1'
; w2 enc 'BR =come_back_here'
STR w1, [victim]
STR w2, [victim, #4]
IC IVAC victim
STR w0, [victim]
RET victim
come_back_here:
```

Figure 4: STORE+RET: If victim cached, runs old code (x3=1), else runs new (x3=0).

```
LDRB x0, [victim]
STR  x1, [victim]
```

Figure 5: SPLIT-STORE: First partial store completes, second faults → bytes stored left if cached.

```
LDR x0, [victim]
STR x0, [page-boundary-ptr]
```

Figure 6: TRANSLATION-RACE: Partial store succeeds before fault when victim cached.

```
LDR x0, [victim] ; [victim]=0x3fff
LDR x1, [x0]
```

Figure 7: POINTER-CHASE: Faults at `0x3fff` (victim cached) vs. `0x4000` (victim uncached).

Demo: Leaking Touch Input on Pixel 9