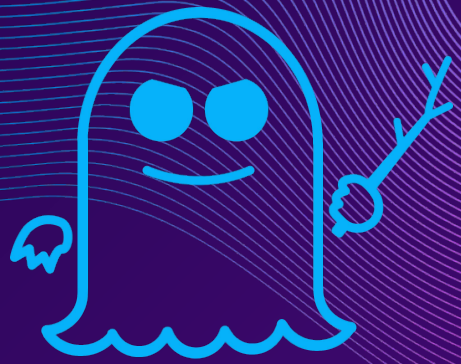
SIDE-CHANNEL IN 15 MINUTES

Cache, Timing, and Essential Defences

WHAT IS A SIDE CHANNEL ATTACK?

Exploiting indirect signals to extract sensitive information

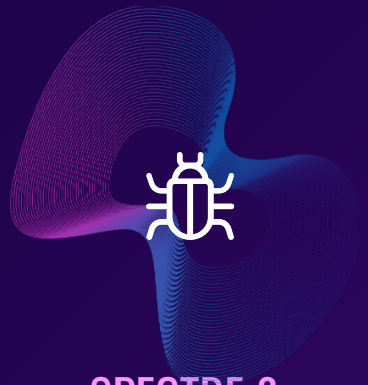
- Execution time
- Power Consumption
- Cache behaviour



SPECTRE

MELTDOWN

REAL-WORLD SIDE-CHANNEL ATTACKS



SPECTRE & MELTDOWN

Speculative execution to leak sensitive data

SMART CARDS

Power analysis to extract cryptographic keys

XBOX 360

Time measurement to retrieve CPU keys

Flush+Reload: A Cache-Based Side-Channel

Invisible Access Patterns Revealed Through Timing

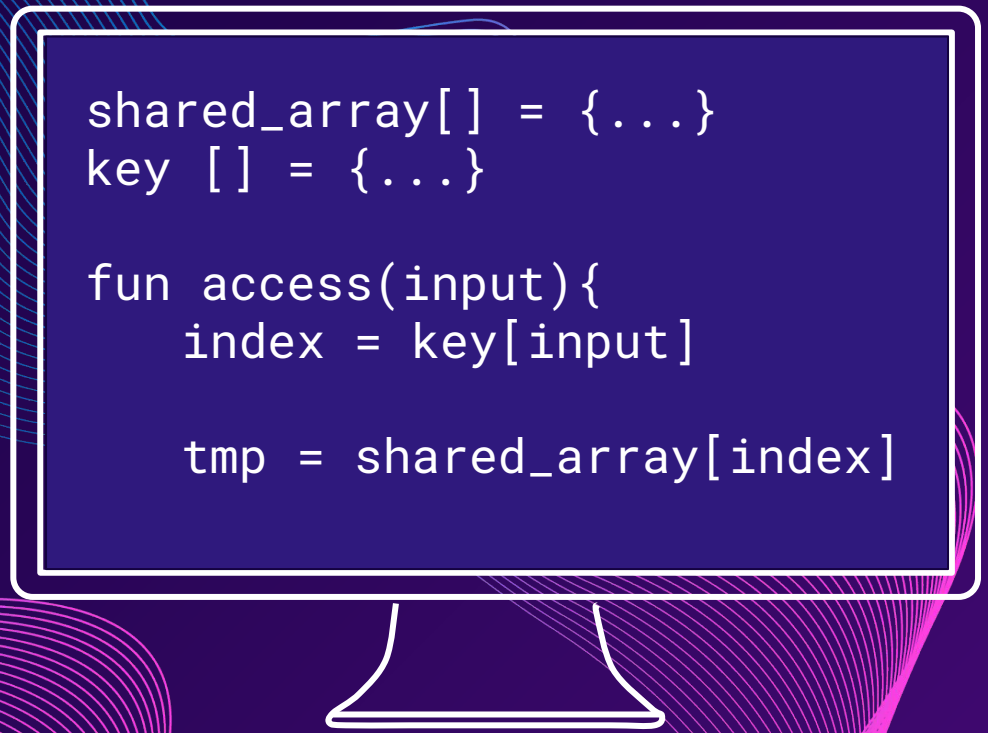
CONCEPT

- Flushes specific cache lines shared with the victim
- Victim's normal execution triggers a reload, creating a timing footprint.
- By measuring reload times, the attacker infers which data the victim accessed.

WHY IT MATTERS

- Minimal privileges if memory is shared (e.g., shared libraries)
- High precision: can reveal cryptographic keys or other sensitive info
- Often effective in multi-tenant or virtualized environments

Flush+Reload: the victim



```
shared_array[] = {...}
key [] = {...}

fun access(input){
    index = key[input]

    tmp = shared_array[index]
```

- **Shared memory usage**: Victim accesses shared_array[]
- **Secret index**: index = key[input]
- **Cache line load**: The CPU brings the relevant line into cache

Flush+Reload: the attacker

- **Flush**: Invalidate shared cache lines (CLFLUSH)
- **Trigger**: Let the victim run access(input)
- **Reload & Time**: Measure access time on each element in shared_array[]
- **Inference**: Fast access ⇒ Victim reloaded that line ⇒ Secret index exposed

```
CLFLUSH(shared_array)
victim.access(some_value)

for i in len(shared_array):
    start_time = time()
    tmp = shared_array[i]
    end = time() - start_time
```

THE == OPERATOR

In most languages, **==** is a byte-wise operation which returns False when the first mismatch occurs

```
def equals(str1, str2, n):
    for i in range(n):
        if str1[i] != str2[i]:
            return False
    return True
```

Timing attack on ==

```
password = ‘AAAAAA’

start = time()
check_password()

if time() - start < par:
    #n chars are correct
```

IMPACT AND MITIGATIONS

KEY LEAKS

Recovery of keys and passwords

DATA EXFILTRATION

Sensitive info can be read

PRIVILEGE ESCALATION

Malicious code bypasses security boundaries

CONSTANT-TIME CODE

Avoid early exits

ISOLATE RESOURCES

Restrict memory sharing

HW/SW MITIGATIONS

Enable CPU/OS feature

THANKS FOR YOUR ATTENTION!

@genricoloni