Towards Faster Speculative Execution Attacks

Damen Maughan
B.S. Computer Science
B.S. Applied Computational Math

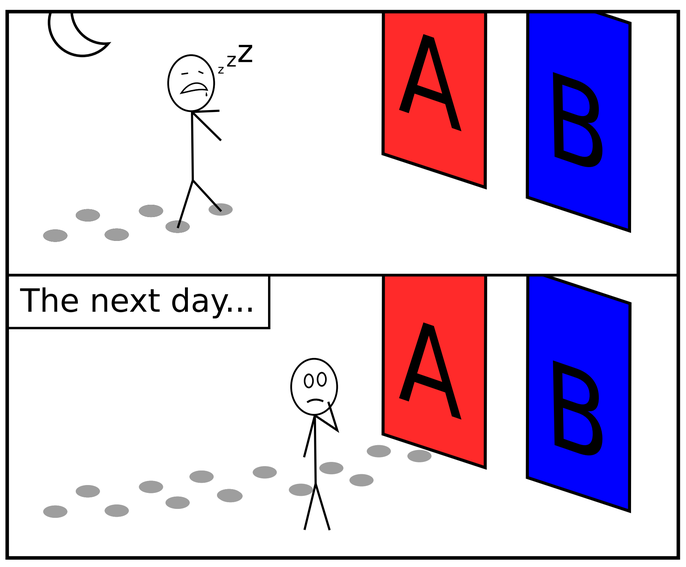
Advisor: Ren Quinn - CS Department

Background - Spectre

"Spectre Attacks: Exploiting Speculative Execution", Paul Kocher et. al.

- A vulnerability in modern processors (namely x86) is described, called "Spectre".
- It takes advantage of a processor's speculative execution.
- Malicious, transient instructions are speculatively executed.
- These instructions are reverted by the processor. However, they have a noticeable effect on the cache state of the CPU.

Using Spectre, touch some secret data. Then, look back at your "footprints" to figure out what it was.

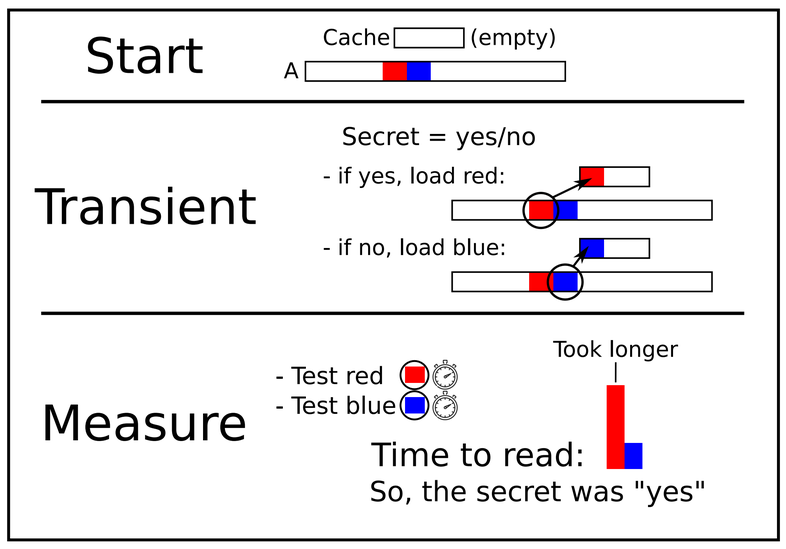


Even if we don't remember, we can still trace our steps.

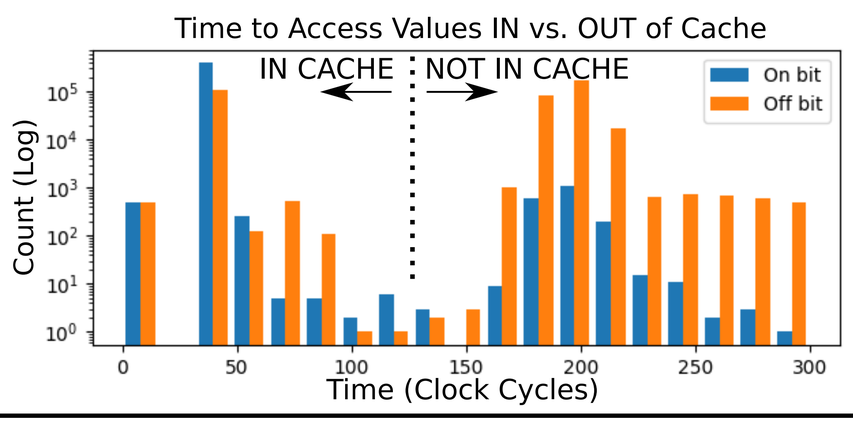
Background - Cache Attacks

Anatomy of a FLUSH + RELOAD Cache Attack:

- Start with nothing in cache
- Access secret data, touch cache
- CPU takes the secret data away
- Check cache to see what it was



- Initially, we FLUSH an array that we control out of the cache.
- Using the Spectre vulnerability, access a secret value through speculative execution.
- Access a position in our array using that value as an index. This will LOAD it into cache.
- At this point, the state will be reset and we will lose access to our secret value.
- Then, time how long it takes to access each index in the array.
- The position that responds the fastest corresponds with the secret value.5



SPECTRE STRATEGY (OLD)

```
A = Array of size 256
s = secret value (1 byte)

victim_function(x):
    if x is valid:
        return s

v = victim_function(x)
A[v]

check all indexes A[i]

slowest index i = v
```

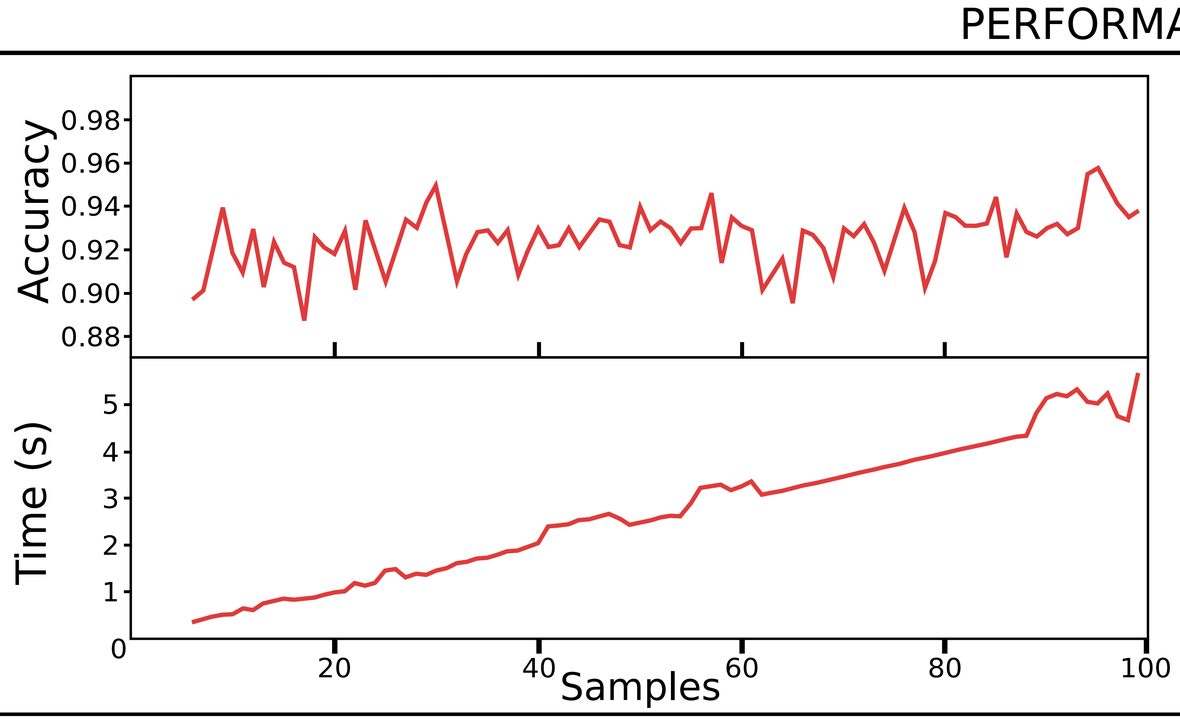
- 256 possible values
- The conditional branch which we are exploting
- If valid, return secret. However, s is returned speculatively anyway
- Grab secret value
- "Write it down" by reading A[v] into cache
- Now, CPU makes us forget what it was.
- Time how long it takes to access each index (each possible value)
- When one of them takes longer than the others, that position corresponds to our secret value!

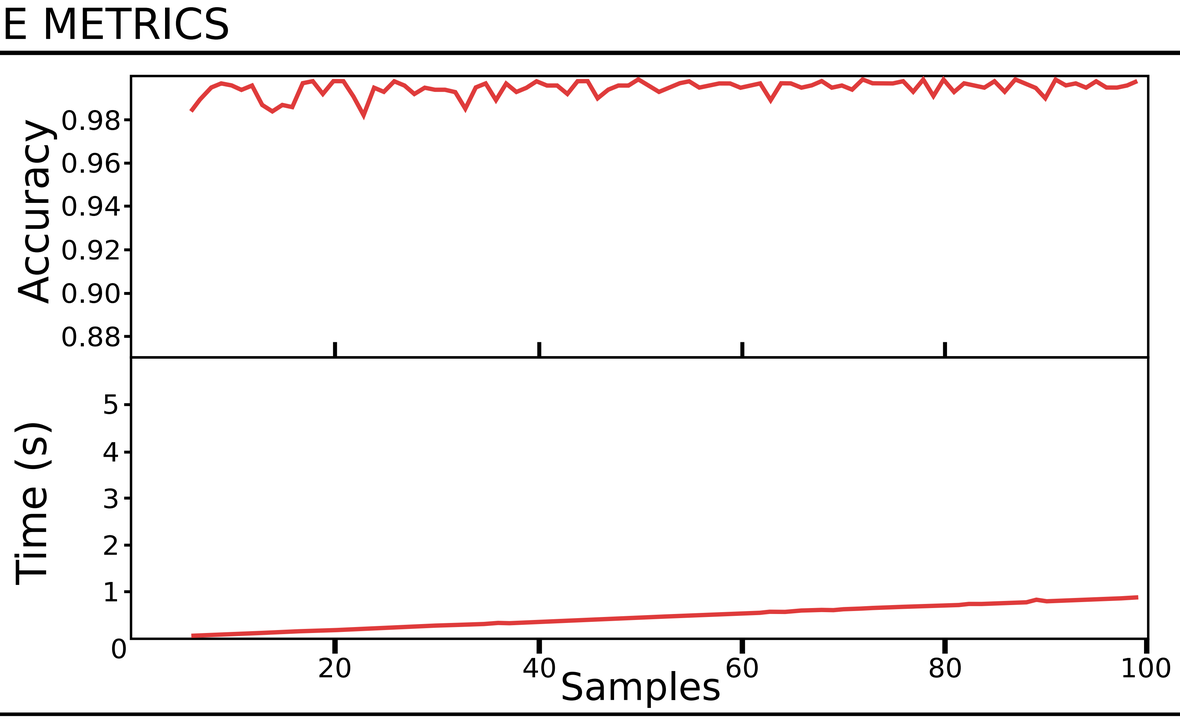
OPTIMIZED STRATEGY (NEW)

```
A = Array of size 2*8
s = secret value (1 byte)

victim_function(x):
    if x is valid:
        return s

v = victim_function(x)
for bit in v:
    A[bit_i][bit]

for each A_b in A:
    check each position A_b[i]

for each bit in V:
    slowest A_b = bit
```

- 8 arrays, 2 possible values each
- Same victim function and exploit
- Use the individual bits of the secret value to read 8 positions into cache
- Now, CPU makes us forget what it was.
- Then, time all 8 arrays
- Slowest value in each array becomes that bit's value
- Re-assemble bits into secret value!

PERFORMANCE METRICS





The method used in Spectre is functional, and gets their point across in an understandable manner. As a cost for that simplicity, it is rather slow.

By reading individual bits, we significantly reduce the amount of cache misses, and as such, increase the overall speed of data reading. As a cost for this speed, it is quite complex.