# CS 578: CYBER-SECURITY

# PART III: SIDE-CHANNELS

Sanghyun Hong
sanghyun.hong@oregonstate.edu


SAIL
Secure AI Systems Lab

## How can we break the isolation?

- Rowhammer breaks **integrity**
- Side-channels break **confidentiality**

# SPECTRE

# PRELIMINARIES ON SPECULATIVE EXECUTION

- Speculative execution is a CPU optimization
  - Instruction cycle: fetch – decode – execute
  - Instruction pipeline: instruction-level parallelism (on a single CPU)

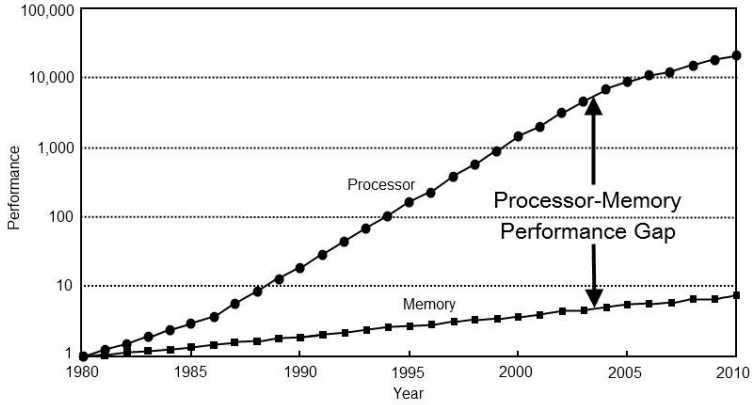
| Instr. No. | Pipeline Stage | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EX | MEM | WB | | |
| 2 | | IF | ID | EX | MEM | WB | |
| 3 | | | IF | ID | EX | MEM | WB |
| 4 | | | | IF | ID | EX | MEM |
| 5 | | | | | IF | ID | EX |
| Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

# PRELIMINARIES ON SPECULATIVE EXECUTION

- Speculative execution is a CPU optimization
  - Out-of-order execution for speed-ups
  - Use to reduce the cost of, *e.g.*, conditional branch

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

- The first line causes a delay until *x* arrives from the memory
- The time it takes to load *x* from memory needs more cycles than running instructions
- A naïve solution is to *wait…*
  but do we have a better solution?

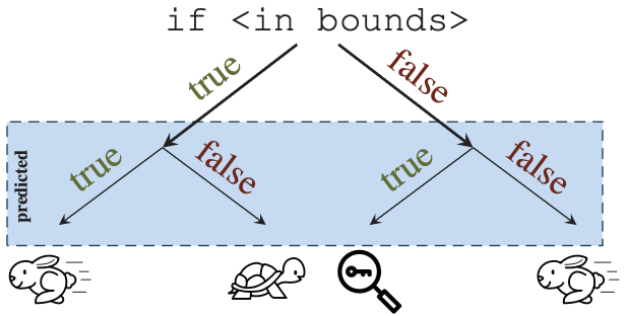


# PRELIMINARIES ON SPECULATIVE EXECUTION

- Speculative execution is a CPU optimization
  - Out-of-order execution for speed-ups
  - Use to reduce the cost of, *e.g.*, conditional branch

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  - The first line causes a delay until *x* arrives from the memory
  - The time it takes to load *x* from memory needs more cycles than running instructions
  - Run the next instructions in the instruction pipeline



# PRELIMINARIES ON SPECULATIVE EXECUTION

- Speculative execution is a CPU optimization
  - Out-of-order execution for speed-ups
  - Use to reduce the cost of, *e.g.*, conditional branch

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  - The first line causes a delay until *x* arrives from the memory
  - The time it takes to load *x* from memory needs more cycles than running instructions
  - Run the next instructions in the instruction pipeline
    - If the *x* satisfies the "if" condition, then commit – performance gain
    - Otherwise, discard the faulty work

# PRELIMINARIES ON SPECULATIVE EXECUTION

- Speculative execution is a CPU optimization
  - Out-of-order execution for speed-ups
  - Use to reduce the cost of, *e.g.*, conditional branch

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  - The first line causes a delay until *x* arrives from the memory
  - The time it takes to load *x* from memory needs more cycles than running instructions
  - Run the next instructions in the instruction pipeline
    - If the *x* satisfies the "if" condition, then commit – performance gain
    - Otherwise, discard the faulty work
  - CPU makes its errors on its on!

# SPECTRE ATTACK (VARIANT 1) – CONDITIONAL BRANCH MISPREDICTION

- Attack scenario

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  - The above code runs in secure environments
  - The attacker wants to read the memory
  - The attacker controls the variable *x*
  - *array1_size* and *array2* is not in cache
  - Suppose the memory status is like the left figure
    - The *array1_size* is 8 bytes

**Memory & Cache Status**

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[… lots of memory up to `array1` base+N…]
`09 F1 98 CC 90...` (something secret)

`array2[ 0*512]`
`array2[ 1*512]`
`array2[ 2*512]`
`array2[ 3*512]`
`array2[ 4*512]`
`array2[ 5*512]`
`array2[ 6*512]`
`array2[ 7*512]`
`array2[ 8*512]`
`array2[ 9*512]`
`array2[10*512]`
`array2[11*512]`
`...`

Contents don't matter
only care about cache ***status***

Uncached Cached

# SPECTRE ATTACK (VARIANT 1) – CONDITIONAL BRANCH MISPREDICTION

- Attack scenario

```
if (x < array1_size)
    y = array2[array1[x] * 4096];
```

  - The variable *x* (control) is set to > 8 bytes
  - CPU runs speculative execution as if "if" is true
  - CPU reads the address *array1* base + *x*
    - It returns the secret byte = 09 (fast – in cache)
    - Requests memory at (*array2* base + 09 * 4096)
    - Brings *array2[09*4096]* into cache
    - Realize the "if" statement is false, then discard this work
  - The control returns to the caller
  - The attacker uses cache side-channels to read 09

**Memory & Cache Status**

`array1_size = 00000008`

Memory at `array1` base address:
8 bytes of data (value doesn't matter)
[… lots of memory up to `array1` base+N…]
`09 F1 98 CC 90...` (something secret)

`array2[ 0*512]`
`array2[ 1*512]`
`array2[ 2*512]`
`array2[ 3*512]`
`array2[ 4*512]`
`array2[ 5*512]`
`array2[ 6*512]`
`array2[ 7*512]`
`array2[ 8*512]`
`array2[ 9*512]`
`array2[10*512]`
`array2[11*512]`
`...`

Contents don't matter
only care about cache ***status***

Uncached | Cached

# SPECTRE ATTACK (VARIANT 2) – POISONING INDIRECT BRANCHES

- Branch predictor
  - Every 5-7 instructions of a program has a branch (a lot!)
  - Costly
    - If the jump address is in a cache – fast
    - If the jump address is not in a cache – slow, wait for the address to come from memory
  - Consider an example C program below

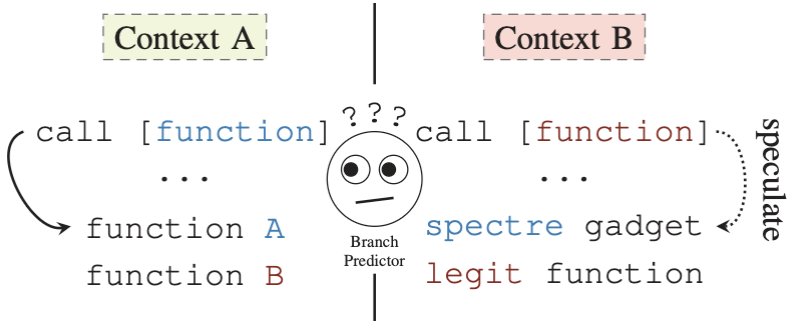
```
for(i=0 ; i < m ; i++)
    for(j=0; j<n ; j++)
        begin S1; S2; …; Sk end;
```

# SPECTRE ATTACK (VARIANT 2) – POISONING INDIRECT BRANCHES

- Branch predictor
  - Branch predictor presumably jumps to a predicted address
    - Based on the branch history (a collection of previous jump addresses)
      - On an Intel Haswell, ~29 prior addresses are used
      - On an AMD Ryzen, ~9 prior branches are used
    - Run a jump
      - If the memory address is the correct one – commit
      - If the address is incorrect – discard faulty work

# SPECTRE ATTACK (VARIANT 2) – POISONING INDIRECT BRANCHES

- Branch predictor
  - Branch predictor presumably jumps to a predicted address
    - Based on the branch history (a collection of previous jump addresses)
      - On an Intel Haswell, ~29 prior addresses are used
      - On an AMD Ryzen, ~9 prior branches are used
    - Run a jump
      - If the memory address is the correct one – commit
      - If the address is incorrect – discard faulty work
  - But what if it jumps to the address, it should not to?



# SPECTRE ATTACK (VARIANT 2) – POISONING INDIRECT BRANCHES

- Attack scenario

```
adc  edi,dword ptr [ebx+edx+13BE13BDh]
adc  dl,byte ptr [edi]
```

  - sleep() function is done with $ebx and $edi
  - The attacker controls $ebx and $edi, and they know $edx
  - The attacker sets $edi to the base address of of the probe array *m*
  - The attacker, for example, sets it to "m – 0x13BE13BD – edx"
  - The instruction in the second line will load *m* into the cache
  - Then they do the same cache side-channel to probe the content

# SPECTRE ATTACK

- Mitigations
  - Disable speculative operations (lfense instruction)
  - Prevent access to sensitive (or secret) data
  - Prevent data from entering covert channels
  - Limit data extraction from covert channels
  - Prevent branch poisoning (retpolines[1])

https://support.google.com/faqs/answer/7625886

# Thank You!

Sanghyun Hong

https://secure-ai.systems/courses/Sec-Grad/current


SAIL
Secure AI Systems Lab