# SegScope: Probing Fine-grained Interrupts via Architectural Footprints

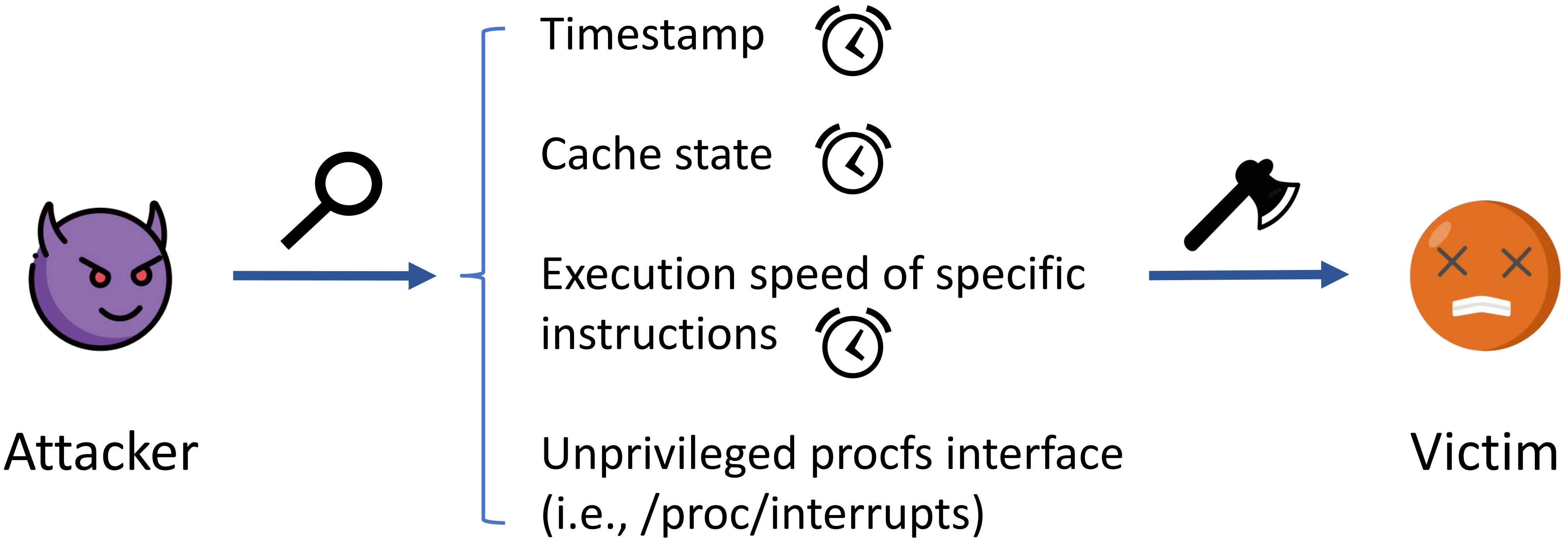
**Xin Zhang**[1], Zhi Zhang[1], Qingni Shen*,

Wenhao Wang, Yansong Gao, Zhuoxi Yang, Jiliang Zhang

[1]Both are joint first author *Corresponding author

# Motivation

Probing interrupts is crucial for interrupt side channel attacks



Attacker

- Timestamp ⏰
- Cache state ⏰
- Execution speed of specific instructions ⏰
- Unprivileged procfs interface (i.e., /proc/interrupts)

Victim

# Timer-constrained Scenario

- To constrain the use of architectural timers, there has been many countermeasures that either detect timers or disable them

- procfs-based probing can be easily defeated by removing unprivileged access to the procfs interface.

# Research Question

- Is there a microarchitectural technique across x86 CPUs probing interrupts without any timers?

- If yes, what attacks can be mounted and what information can be leaked?

# Our work

- We propose SegScope, a new technique that abuses *segment protection* on x86 to acquire fine-grained interrupt observations without relying on any (external) timer

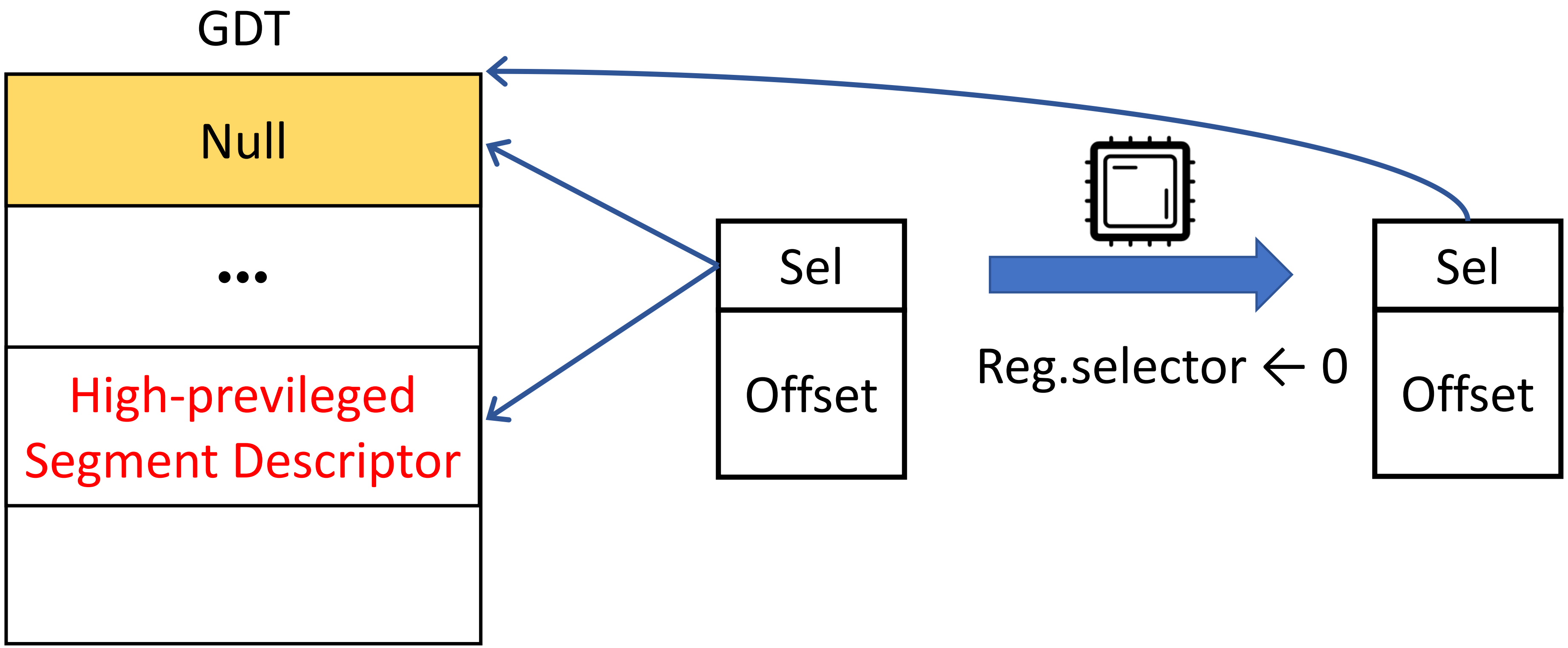
- Our key observation is that some non-zero selector values are considered as *null segment selector* and will be cleared by CPUs when an interrupt occurs

# Segment Protection

- Supported by a wide range of Intel and AMD-based CPUs
- Ensure a user process cannot access the kernel address *outside* the controlled and well-defined interfaces
- CPU *clears* the data segment registers if they contain high-privileged information when returning to the lower-privileged level

# Segment Protection

The x86 architecture defines segment registers for memory segmentation, dividing main memory into segments or sections.



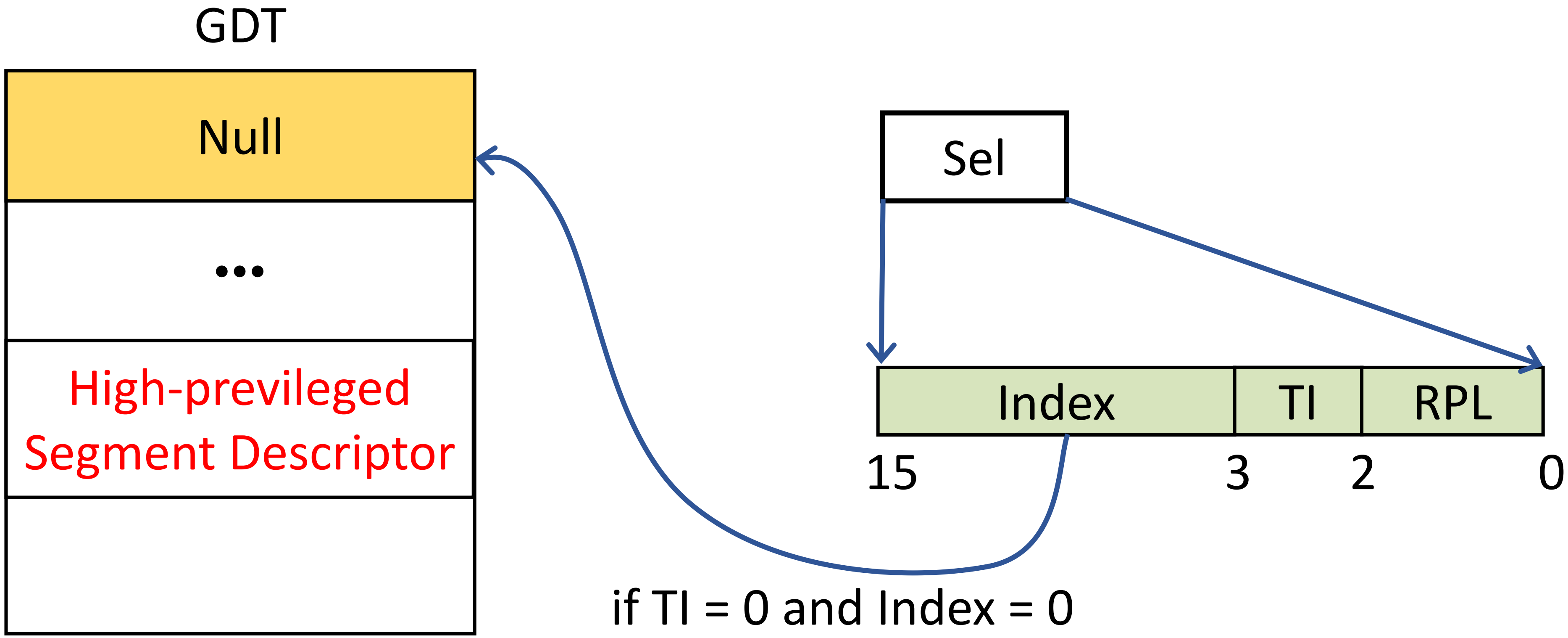
# SegScope

- Segment selectors are stored in the “visible part” of data segment registers (i.e., DS, ES, GS, and FS), which can be read and written by *an unprivileged process*

- Segment selectors can be set to NULL *without any privilege check*. No exception will occur until they are referenced

# Segment Protection

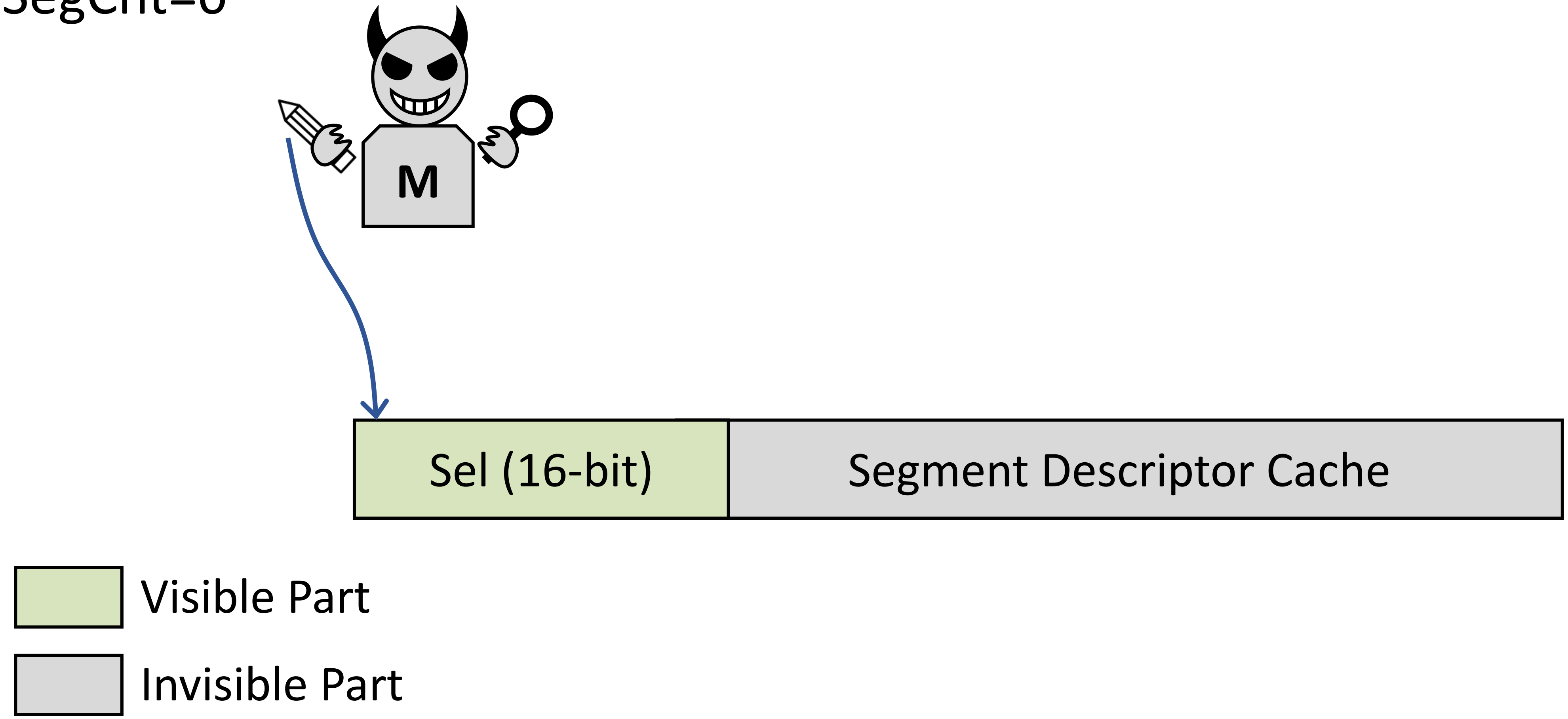
We observe that the NULL segment selector is not always 0 ...



# SegScope

① Initialize Sel=0x0001, 0x0002, or 0x0003;

SegCnt=0



Sel (16-bit)

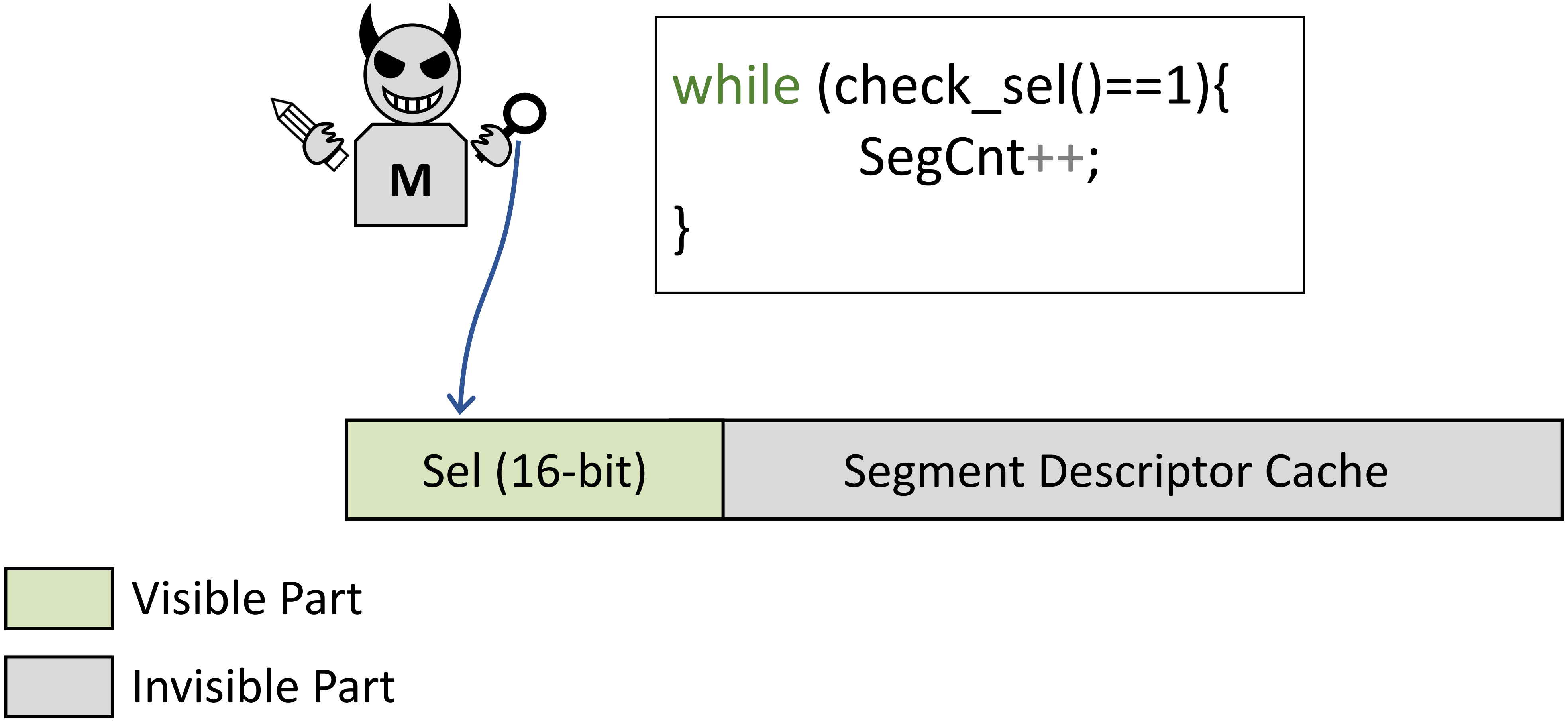
Segment Descriptor Cache

🟩 Visible Part

⬜ Invisible Part

# SegScope

② Loop to check Sel and increment SegCnt



```
while (check_sel()==1){
        SegCnt++;
}
```

Sel (16-bit)

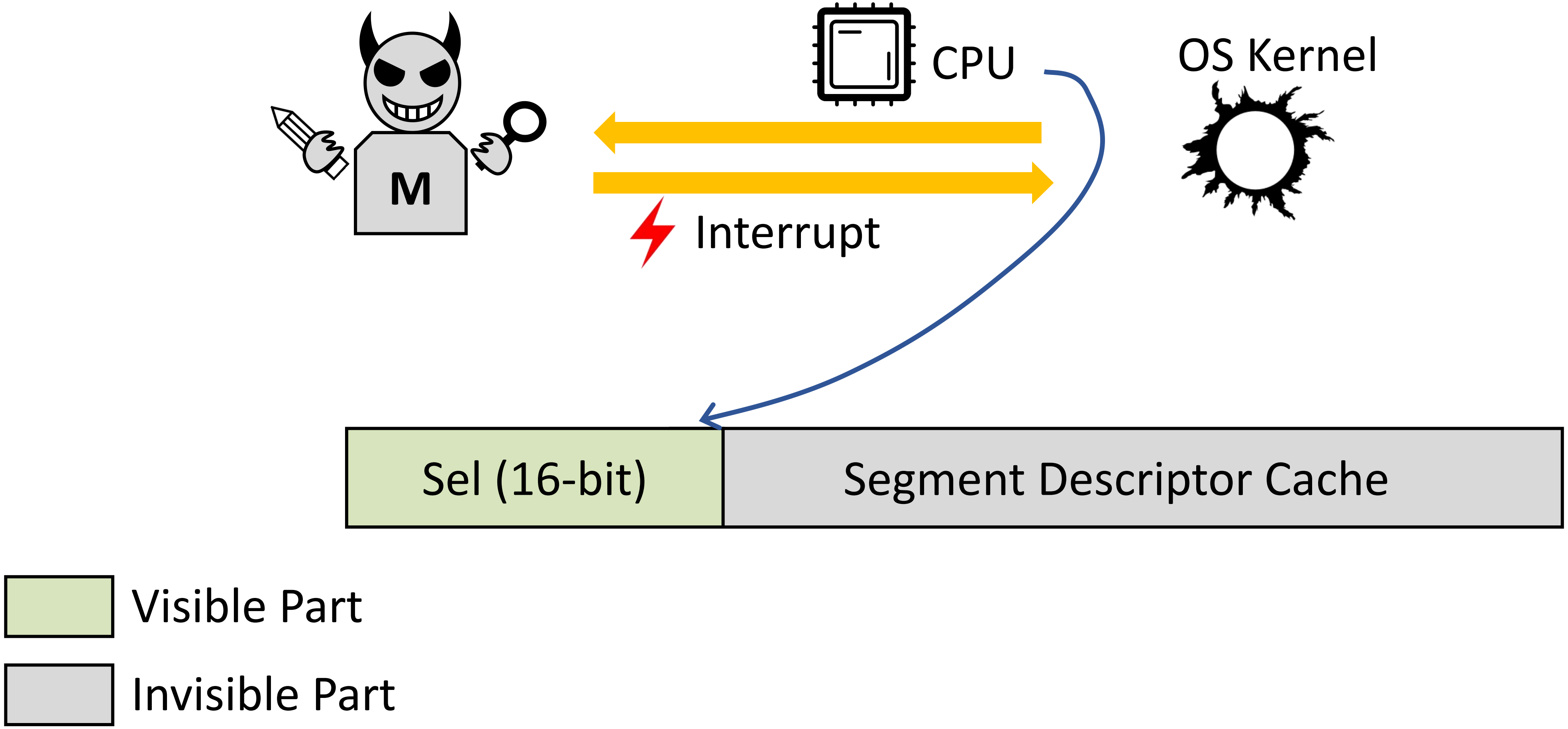
Segment Descriptor Cache

🟩 Visible Part

⬜ Invisible Part

# SegScope

③ CPU will clear the null segment selector to 0



M

CPU

OS Kernel

Interrupt

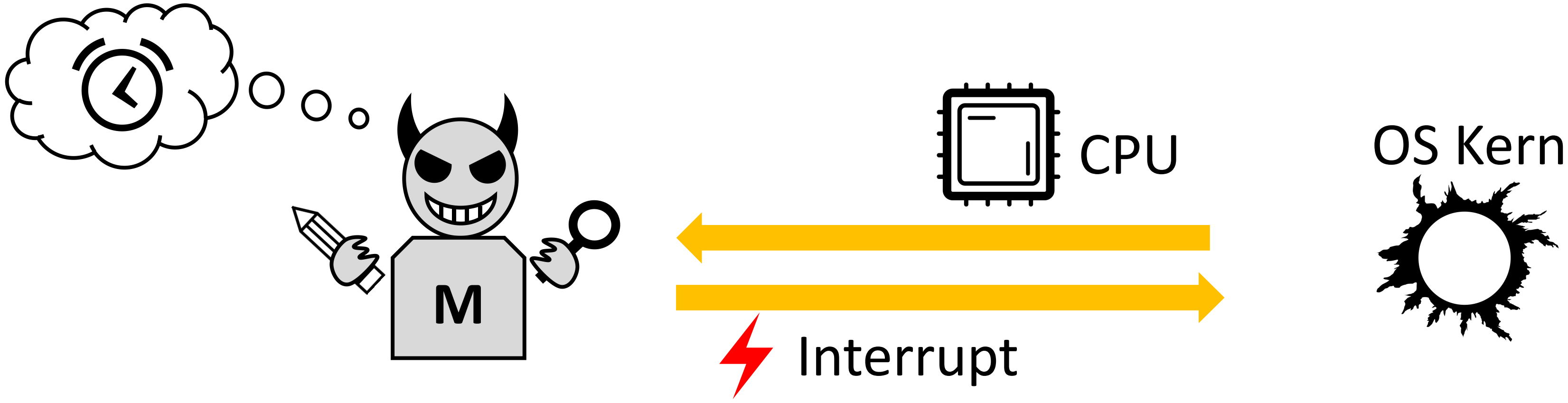
| Sel (16-bit) | Segment Descriptor Cache |
|---|---|

🟩 Visible Part

⬜ Invisible Part

# SegScope

④ Stop counting and break the loop



| | |
|---|---|
| Sel (16-bit) | Segment Descriptor Cache |

🟩 Visible Part

⬜ Invisible Part

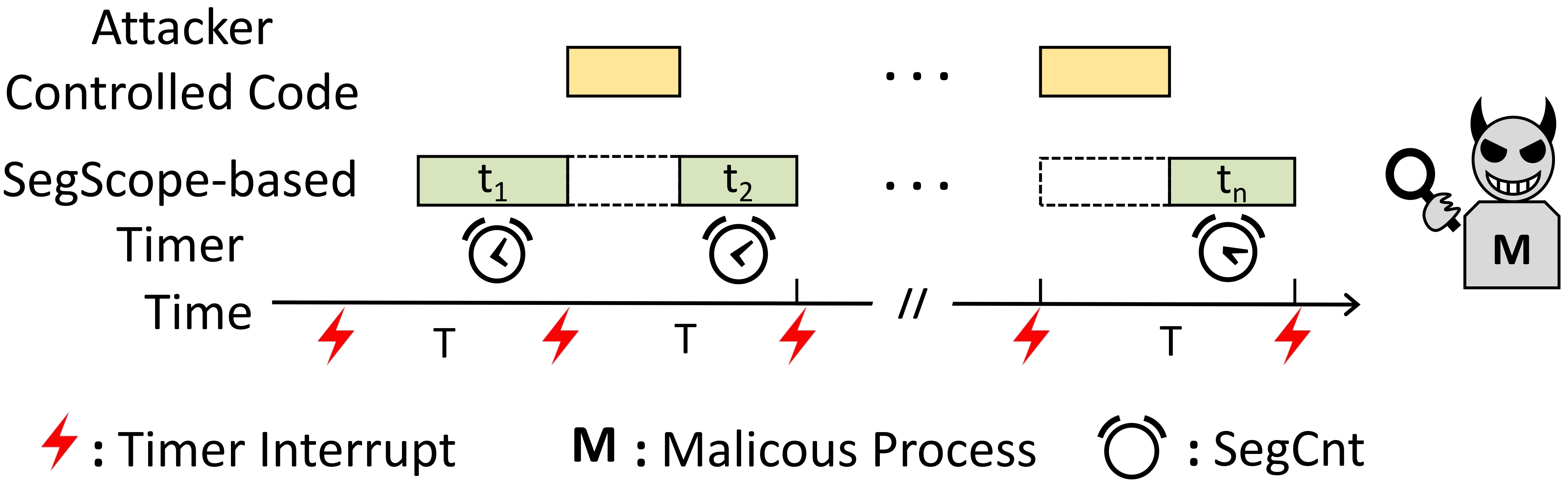
# SegScope-based Timer

- On x86, timer interrupts are generated by Advanced Programmable Interrupt Controller (APIC) at fixed time intervals
- The number of timer interrupts accounts for over 99% of the overall interrupts
- Existing timer interrupt based works assume a privileged user who controls the frequency of timer interrupts, in either attack or defense scenarios

# SegScope-based Timer

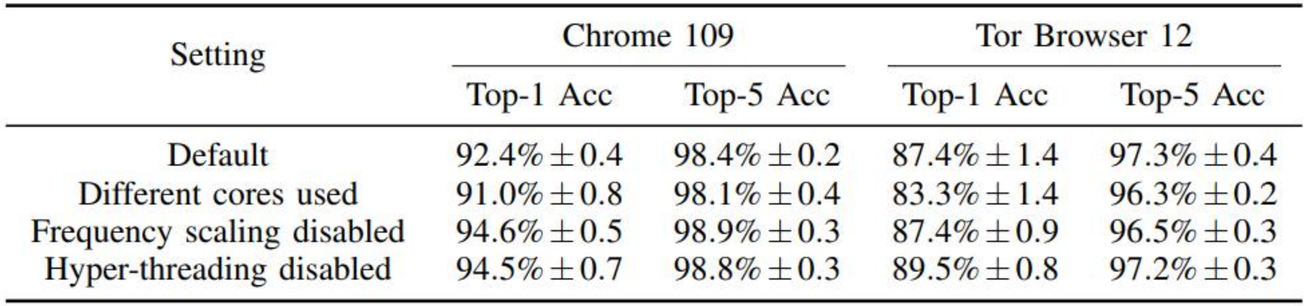
As the time interval between two consecutive timer interrupts is fixed,

SegScope can time the other piece of code that shares the time interval

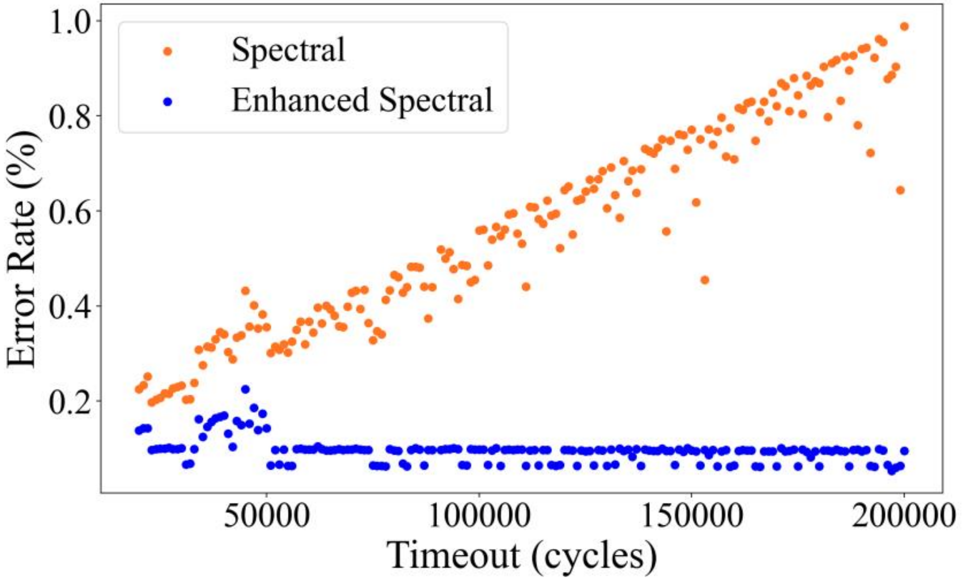


# Case Studies—For SegScope

- We can fingerprint websites with an accuracy of over 90%
- SegScope can filter out the interrupt noise for Spectral, reducing its error rate by 56x

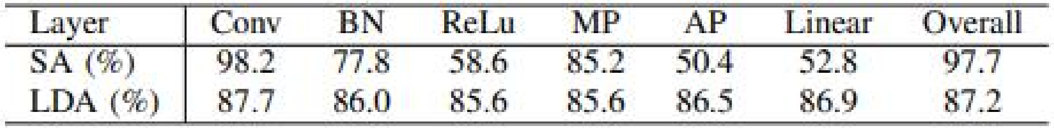


| Setting | Chrome 109 | | Tor Browser 12 | |
|---|---|---|---|---|
| | Top-1 Acc | Top-5 Acc | Top-1 Acc | Top-5 Acc |
| Default | 92.4% ± 0.4 | 98.4% ± 0.2 | 87.4% ± 1.4 | 97.3% ± 0.4 |
| Different cores used | 91.0% ± 0.8 | 98.1% ± 0.4 | 83.3% ± 1.4 | 96.3% ± 0.2 |
| Frequency scaling disabled | 94.6% ± 0.5 | 98.9% ± 0.3 | 87.4% ± 0.9 | 96.5% ± 0.3 |
| Hyper-threading disabled | 94.5% ± 0.7 | 98.8% ± 0.3 | 89.5% ± 0.8 | 97.2% ± 0.3 |

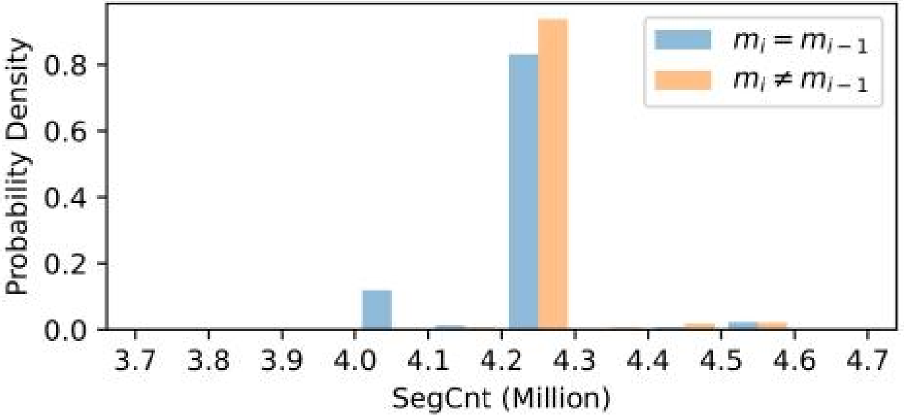


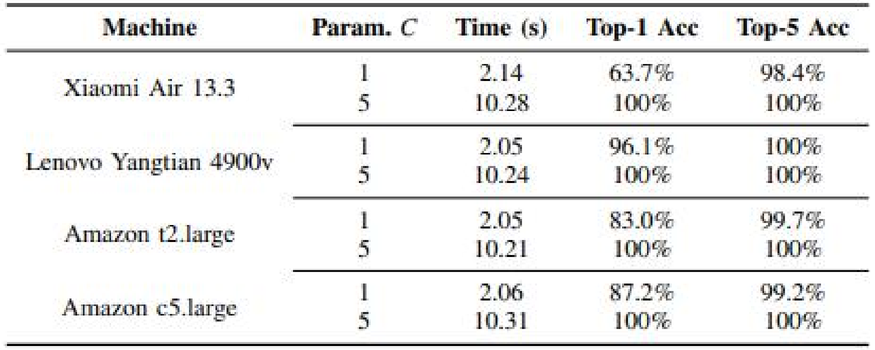
# Case Studies—For SegScope-based Timer

- We can steal DNN model architectures and extract SIKE keys.
- We can break KASLR within 10 seconds and mount Flush+Reload based Spectre attack

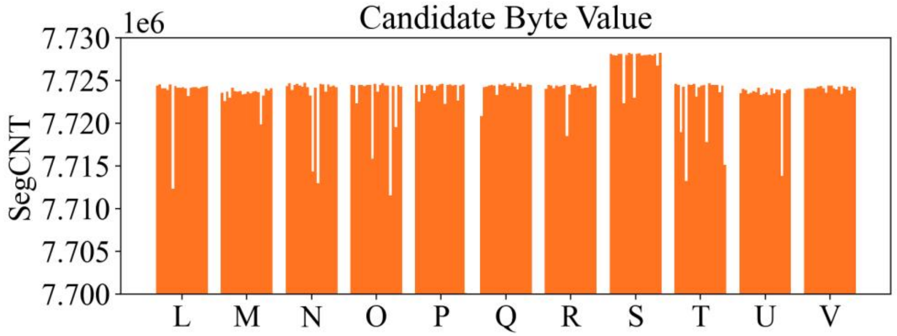


| Layer | Conv | BN | ReLu | MP | AP | Linear | Overall |
|---|---|---|---|---|---|---|---|
| SA (%) | 98.2 | 77.8 | 58.6 | 85.2 | 50.4 | 52.8 | 97.7 |
| LDA (%) | 87.7 | 86.0 | 85.6 | 85.6 | 86.5 | 86.9 | 87.2 |





| Machine | Param. $C$ | Time (s) | Top-1 Acc | Top-5 Acc |
|---|---|---|---|---|
| Xiaomi Air 13.3 | 1 | 2.14 | 63.7% | 98.4% |
| | 5 | 10.28 | 100% | 100% |
| Lenovo Yangtian 4900v | 1 | 2.05 | 96.1% | 100% |
| | 5 | 10.24 | 100% | 100% |
| Amazon t2.large | 1 | 2.05 | 83.0% | 99.7% |
| | 5 | 10.21 | 100% | 100% |
| Amazon c5.large | 1 | 2.06 | 87.2% | 99.2% |
| | 5 | 10.31 | 100% | 100% |



# Mitigations

- Software Mitigation: Modifying OS kernel/x86 CPU architecture
- Hardware Mitigation: A potential strategy of mitigating SegScope is to keep the values of the segment registers unchanged in future architectures, which however introduces a new covert channel
- Hardware-software Co-design: when context switch occurs, OS kernels save and restore the segment registers for every process, and CPUs preserve the non-zero segment selectors as-is.

# Mitigations

- Software Mitigation: Modifying OS kernel/x86 CPU architecture
- Hardware Mitigation: A potential strategy of mitigating SegScope is to keep the values of the segment registers unchanged in future architectures, which however introduces a new covert channel
- Hardware-software Co-design: when context switch occurs, OS kernels save and restore the segment registers for every process, and CPUs preserve the non-zero segment selectors as-is.

# Mitigations

- Software Mitigation: Modifying OS kernel/x86 CPU architecture
- Hardware Mitigation: A potential strategy of mitigating SegScope is to keep the values of the segment registers unchanged in future architectures, which however introduces a new covert channel
- Hardware-software Co-design: when context switch occurs, OS kernels save and restore the segment registers for every process, and CPUs preserve the non-zero segment selectors as-is.

# Takeaway

- A general “µarch-state-to-arch-state” converter via observing *architectural footprints* on x86
- SegScope can be used to probe fine-grained interrupts *without any (external) timers*
- Based on SegScope, timer interrupts can be exploited by *unprivileged attackers* to build a new fine-grained timer.
- Artifact: https://github/zhangxin00/segscope

# SegScope: Probing Fine-grained Interrupts via Architectural Footprints

# Q & A