Week 1 Report

Ben Chen

Dept of Computer Science and Engineering, SUSTech

September 12, 2024

Cross-Core Interrupt Detection[1]

Motivation:

- To accelerate inter-process communication, Intel introduces user-triggered interrupt instruction in latest Xeon processor, which result 4 times faster than tradition interrupts.

Cross-Core Interrupt Detection[1]

Motivation:

- To accelerate inter-process communication, Intel introduces user-triggered interrupt instruction in latest Xeon processor, which result 4 times faster than tradition interrupts.
- Past attacks exploiting interrupts are infeasible.

Cross-Core Interrupt Detection[1]

Motivation:

- To accelerate inter-process communication, Intel introduces user-triggered interrupt instruction in latest Xeon processor, which result 4 times faster than tradition interrupts.
- Past attacks exploiting interrupts are infeasible.
- The unpriviledged user interrupt raises a security concerns: it makes side-channel attacks like keystroke and web fingerprint attack eaiser to conduct.

Cross-Core Interrupt Detection[1]

The interrupt can be sent to any cores including itself

- The interrupt is sent through shared bus $\rightarrow$ other processes' interrupts will lead to a higher latency in IPI

Cross-Core Interrupt Detection[1]

The interrupt can be sent to any cores including itself

- The interrupt is sent through shared bus $\rightarrow$ other processes' interrupts will lead to a higher latency in IPI
- By sending a IPI to themselve, attacker process can know the interrupt behaviours (network, keyboard, etc.) by measuring the propagation of IPI and comparing with benchmark.

Cross-Core Interrupt Detection[1]

The interrupt can be sent to any cores including itself

- The interrupt is sent through shared bus $\rightarrow$ other processes' interrupts will lead to a higher latency in IPI
- By sending a IPI to themselve, attacker process can know the interrupt behaviours (network, keyboard, etc.) by measuring the propagation of IPI and comparing with benchmark.
- The author also measured the virtualized IPI, but with regular IPI.

Cross-Core Interrupt Detection[1]

To conduct experiment, the author design a scenerio(covert channel) and victim process:

- If sender sends 1 to receiver, sender sends self-IPI (occupy the bus)

Cross-Core Interrupt Detection[1]

To conduct experiment, the author design a scenerio(covert channel) and victim process:

- If sender sends 1 to receiver, sender sends self-IPI (occupy the bus)
- If sender sends 0, sender busy waiting (idle)

Cross-Core Interrupt Detection[1]

To conduct experiment, the author design a scenerio(covert channel) and victim process:

- If sender sends 1 to receiver, sender sends self-IPI (occupy the bus)
- If sender sends 0, sender busy waiting (idle)
- The receiver measures the latency of self-IPI

Cross-Core Interrupt Detection[1]

To conduct experiment, the author design a scenerio(covert channel) and victim process:

- If sender sends 1 to receiver, sender sends self-IPI (occupy the bus)
- If sender sends 0, sender busy waiting (idle)
- The receiver measures the latency of self-IPI
- Researchers measured with native IPI and virtualized IPI, and measured the bit error ratio versus side-channel capacity.

Cross-Core Interrupt Detection[1]

To conduct experiment, the author design a scenerio(covert channel) and victim process:

- If sender sends 1 to receiver, sender sends self-IPI (occupy the bus)
- If sender sends 0, sender busy waiting (idle)
- The receiver measures the latency of self-IPI
- Researchers measured with native IPI and virtualized IPI, and measured the bit error ratio versus side-channel capacity.
- Also tested attacks under other cores stressed (busy with IO), the results shows that the error ratio remains.

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Keystroke**

- Keep running self-IPI and measure using a high precision timer with `rdtsc`

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Keystroke**

- Keep running self-IPI and measure using a high precision timer with `rdtsc`
- To reduce noise, dynamically adjust the base time

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Keystroke**

- Keep running self-IPI and measure using a high precision timer with `rdtsc`
- To reduce noise, dynamically adjust the base time
- Not focusing on text recovering (since data processing needs complex analysis)

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Web Fingerprint**

- Setup: for native, attacker has access to `rdtsc` but not `/proc/interrupts`; for VM, the attacker and victim run on the same host VM.

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Web Fingerprint**

- Setup: for native, attacker has access to `rdtsc` but not `/proc/interrupts`; for VM, the attacker and victim run on the same host VM.
- Evaluted on 100 websites in both close-world and open-world.

Cross-Core Interrupt Detection[1]

Case studies (real-world attack surfaces): **Web Fingerprint**

- Setup: for native, attacker has access to `rdtsc` but not `/proc/interrupts`; for VM, the attacker and victim run on the same host VM.
- Evaluted on 100 websites in both close-world and open-world.
- Also performed trace analyzing with CNN $\rightarrow$ knows what website the victim is viewing

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer
- Modify timer to be secret-independent

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer
- Modify timer to be secret-independent
- Introduces noise by IPI or so

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer
- Modify timer to be secret-independent
- Introduces noise by IPI or so
- General methods: detect abnormally frequent self-IPI

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer
- Modify timer to be secret-independent
- Introduces noise by IPI or so
- General methods: detect abnormally frequent self-IPI
- TLB, Prime+Probe, flush caches...

Cross-Core Interrupt Detection[1]

Mitigation and other attacks using IPI

- Restricted timer $\Rightarrow$ useless since attacker can construct timer
- Modify timer to be secret-independent
- Introduces noise by IPI or so
- General methods: detect abnormally frequent self-IPI
- TLB, Prime+Probe, flush caches...
- Move computing to TEE $\Rightarrow$ malicious host or guest

Real-Time Side Channel Detection[2]

Previous mitigation:

- `fence` instruction $\Rightarrow$ will ensure the instruction is executed before `fence`, can be used to prevent speculative execution when detected.

Real-Time Side Channel Detection[2]

Previous mitigation:

- `fence` instruction $\Rightarrow$ will ensure the instruction is executed before `fence`, can be used to prevent speculative execution when detected.
- Speculative Load Hardening (HLS), inserts constant instead of real data after trainning with instrcutions to predict the sensitive instruction.

Real-Time Side Channel Detection[2]

Previous mitigation:

- `fence` instruction $\Rightarrow$ will ensure the instruction is executed before `fence`, can be used to prevent speculative execution when detected.
- Speculative Load Hardening (HLS), inserts constant instead of real data after trainning with instrcutions to predict the sensitive instruction.
- This work is to detect rather than prevent.

Real-Time Side Channel Detection[2]

Real-time detection method on BOOM:

- Sampling the cache using HPC on BOOM, and focuing on TLB miss and branch predict miss rate.

Real-Time Side Channel Detection[2]

Real-time detection method on BOOM:

- Sampling the cache using HPC on BOOM, and focuing on TLB miss and branch predict miss rate.
- Train multiple layer perceptron network with collected data set: normal and under spectre attack.

Real-Time Side Channel Detection[2]

Real-time detection method on BOOM:

- Sampling the cache using HPC on BOOM, and focuing on TLB miss and branch predict miss rate.
- Train multiple layer perceptron network with collected data set: normal and under spectre attack.
- Perform real-time detection with the trained model, with 2% overhead.

References I

[1] Fabian Rauscher and Daniel Gruss. "Cross-Core Interrupt Detection: Exploiting User and Virtualized IPIs". English. In: *ACM Conference on Computer and Communications Security (CCS) 2024*. ACM Conference on Computer and Communications Security : CCS 2024, CCS '24 ; Conference date: 14-10-2024 Through 18-10-2024. Oct. 2024. DOI: 10.1145/3658644.3690242.

[2] Anh-Tien Le et al. "A Real-Time Cache Side-Channel attack detection system on RISC-V Out-of-Order processor". In: *IEEE Access* 9 (Jan. 1, 2021), pp. 164597–164612. DOI: 10.1109/access.2021.3134256. URL: https://doi.org/10.1109/access.2021.3134256.