Side Channel Risks in Hardware Trusted Execution Environments (TEEs)

Wenhao Wang (王文浩)
May 2019
About Me

- Research Associate Professor (副研究员) of Institute of Information Engineering, CAS
- Previously worked as a visiting researcher in Indiana University (with Prof. XiaoFeng Wang)
- Research Interests
  - System Security
  - Computer Architectural Security
  - Isolation with Hardware Features
  - Privacy Preserving Computing Technologies
  - Cryptography (esp. Symmetric Cryptanalysis)
- Email: wangwenhao@iie.ac.cn
Scenarios when Hardware TEEs are needed

- Users’ private data are delegated to untrusted (public) cloud servers
- Multi-sources (federated) deep learning training
- Machine-Learning As A Service
- Data sharing of genomic data or big data
Scenarios when Hardware TEEs are needed

- Crypto Techniques
  - FHE, MPC, Searchable Encryption, ZK etc.
  - Extremally High Communication and Computation Overhead

- Hardware Techniques
  - Intel TXT, ARM Trustzone, Intel SGX, AMD SEV etc.
Hardware TEEs – A Review

Secure coprocessors
- XOM
- Aegis
- ARM Trustzone

Intel TXT
- Bastion
- Intel SGX

AMD SME/SEV

Intel TME/MKTME

Excluded: Intel CAT/CET/SMEP/SMAP/VT-x/PT
Intel SGX

- Memory Encryption
- Access Control
- Remote Attestation

Data Owner

Outsourced Computation

Remote Server

Untrusted Platform/OS/VMM

SGX Enclaves

Untrusted Part of App
- Create Enclave
- Call Trusted Function (etc.)

Trusted Part of App
- Execute
- Return

Call Gate

Privileged System Code, OS, VMM, BIOS, SMM, etc.

SSN: 999-84-2611
m8U3bcV%P49Q
Intel SGX

- Enclave memory is stored within the Enclave Page Cache
Security Checks are performed when address translation is loaded into TLB.
What is a side channel?

- Side channels from resources shared crossing multi-domains
The cache holds copies of aligned blocks of $B$ bytes in main memory (blocks).

When a memory access instruction is processed, memory cell is searched in the cache first.

If a cache miss occurs, a full memory block is copied into the appropriate set ($S$ possible sets) into one of the $W$ cache lines.
Side channels – An example (Cache Timing Attacks)
Side channels – An example (Cache Timing Attacks)

1. Completely evict victim data from cache
2. Trigger a victim data access
3. Access attacker memory again and see which cache sets are slow
Side channels – An example (Controlled-channel Attacks)

if (input)
  func1()
  func2()

Page X

Page Y
  func1()

Page Z
  func2()

Page fault sequence X, Y
Page fault sequence X, Z
Side channels – Others?

- **Memory Hierarchy**
  - Data Caching creates fast and slow execution paths, leading to timing differences depending on whether data is in the cache or not

- **Function Unit Contention**
  - Sharing of hardware leads to contention, whether a program can use some hardware leaks information about other programs

- **Stateful Functional Units**
  - Program’s behavior can affect state of the function units (e.g. branching target), and other programs can observe the output (which depends on the state)

- **Variable Instruction Execution Timing**
  - Execution of different instructions or same instruction with different operands takes different amount of time

- **Physical Emanations**
  - Execution of programs affects physical characteristics of the chip, such as thermal changes (e.g. avx512), which can be observed
Can we reduce the interrupts for page based attacks?

- 1. Passive observation over the Access bit of a PTE
Can we reduce the interrupts for page based attacks?

- 2. Measuring the time between accesses to pages
Can we reduce the interrupts for page based attacks?

- 3. Clearing TLB entries from the other Hyper-thread to force a page table walk
Hyper-threading (SMT)

- Hyper-Threading enables new side channel attack surfaces

**Multiprocessor**

- Processor Execution Resources
  - AS

**Hyper-Threading**

- Processor Execution Resources
  - AS
  - AS

AS: architectural state (eax, ebx, control registers, etc.)
Problems with Hyper-Threading
Naïve Solutions do not work

- Simply disabling Hyper-Threading
  - No effective way to verify
    - `cpuid`, `rdtscp` and `rdpid` are not supported in enclave mode
  - Remote attestation
    - Does not contain information about Hyper-Threading (before our work)

- Create a shadow thread from the enclave program to occupy the other hyper-thread

- How to reliably verify the physical-core co-location?
Closing HT-Side Channels on SGX with Contrived Data Races

- Co-location test with Contrived Data Races

- Co-located: Both threads observe data races with high probability
- Otherwise: At least one observe data races with low probability

Cache coherence protocol
(latency ~ 190 cycles on Skylake)

(< 10 cycles)
When co-located, communication time < execution time

Each thread read the value written by the other thread with very high probability.
• When **not** co-located, communication time > execution time
• Each thread read the value written by the other thread with very low probability.
Closing HT-Side Channels on SGX with Contrived Data Races

Use of CMOV instructions

Hypothesis Test based security model

Different padding instruction patterns
HyperRace: An LLVM based tool to eradicate all side-channel threats due to Hyper-Threading.

```
if (aex_detected()) {
  co_location_test();
}
```
Conclusion

- The SGX design opens up many side channels.
- These side channels can be combined
  - To make the attack stealthy and hard to detect
  - To achieve fine-grained observation
- The attacker can even reduce the noises by controlling the SW/HW environment.
- The side channel threats against SGX can not be ignored.

- How to design future TEEs?
  - HW/SW co-design?
  - Real world implications
Questions?

Thank you