Secure Speculative Execution Processors

Ilia Lebedev, Srini Devadas

With contributions from Victor Costan, Vladimir Kiriansky, Saman Amarasinghe and Joel Emer
Outline

• Violating isolation by exploiting speculative execution

• Defenses against cache timing attacks

• Secure enclaves in Intel SGX and MIT Sanctum
Outline

• Violating isolation by exploiting speculative execution

• Defenses against cache timing attacks

• Secure enclaves in Intel SGX and MIT Sanctum
Architectural Isolation

Fundamental to maintaining correctness and privacy!
Control Flow Speculation for Performance

Sequential Instruction Execution
- I: Compute
- I+1: Compute
- I+2: Compute
- I+3: Compute

Non-sequential Instruction Execution
- Correct direction:
  - J: Compute
  - J+1: Compute
  - J+2: Compute

- Mis-speculated direction:
  - K: Compute
  - K+1: Compute
  - K+2: Compute
Control Flow Speculation is insecure

Speculative execution does not affect architectural state → “correct”
... but can be observed via some “side channels” (primarily cache tag state)

... and attacker can influence (mis)speculation → branch predictor inputs not authenticated

A huge, complex attack surface!
Side Channels in the Wild

- Real systems: large, complex, cyberphysical (not secure)

- Spies potentially everywhere
Attack Schema

1. Create a channel
2. Create the transmitter
3. Launch the transmitter
4. Access the secret
Building a Transmitter

Pre-existing (RSA conditional execution example)
Written by attacker (Meltdown)
Synthesized out of existing victim code by attacker (Spectre)
Outline

• Violating isolation by exploiting speculative execution

• Defenses against cache timing attacks

• Secure enclaves in Intel SGX and MIT Sanctum
Here, we focus on this one step:

1. Create a channel
2. Create the transmitter
3. Launch the transmitter
4. Access the secret

Block any of these steps!
Intel’s Cache Allocation Technology (CAT)

CAT can be configured to prevent a potential transmitter from evicting LLC lines of a potential receiver.

Way partitioning is flexible, but CAT is built for QoS and not for security

- Shared addresses are visible across domains
- Replacement metadata updates are not isolated
Intel’s CAT leaks information through cache hits

CAT restricts cache fills to a portion of the LLC

address, class of service

set index

set 0
set 1

way 0
way 1
way w

Fill the selected line, invalidate/writeback current contents, if applicable

No match $\rightarrow$ cache miss $\rightarrow$ cache fill

Match $\rightarrow$ cache hit $\rightarrow$ send/modify data

Cache hits are not isolated across domains
Sharing replacement metadata leaks information.

**Receiver** sets up PLRU metadata: \( \text{LOAD}(A) ; \text{LOAD}(B) \)

- Tree-PLRU metadata
- Transmitter accesses \( T_x \)
- Next victim

**Transmitter** accesses \( T_x \)

- \( T_A \) \( T_B \) \( T_C \) \( T_X \)
- (or doesn’t)

**Receiver** observes PLRU metadata via an eviction:

- \( T_D \) \( T_B \) \( T_C \) \( T_X \)
- \( T_A \) \( T_B \) \( T_C \) ?

**Receiver** probes for \( T_x \)

- \( t = \text{time}() \) \( \text{LOAD}(A) \)
- \( t = \text{time}()-t \)

- t is large
- t is small
Our Work: DAWG: Dynamically Allocated Way Guards

DAWG tracks *global* protection domains

Caches ensure protection domains do not interfere via cache tags or replacement metadata.

Cores *tag each access* with a protection domain id:

Core 1’s DAWG domain_id MSR

<table>
<thead>
<tr>
<th>Instruction fetch domain</th>
<th>Load domain</th>
<th>Store* domain</th>
</tr>
</thead>
</table>

Need DAWG-like approach for other shared microarchitectural state, e.g., branch predictors
Complication!

Masking cache hits may lead to duplicated lines!
→ OS ensures only clean, read-only lines are duplicated.

This is conveniently compatible with modern copy-on-write sharing
- Efficient ways to handle MMAP and Fork
Outline

• Violating isolation by exploiting speculative execution

• Defenses against cache timing attacks

• Secure enclaves in Intel SGX and MIT Sanctum
A Typical Computer System’ TCB

Software...

- (Ring 3)
- App
- App
- Secure App

... Running on hardware

- TRUSTED
- OS Kernel (Ring 0)
- Hypervisor (Ring 0, VMX root)
- BIOS (SMM)
Intel’s SGX to reduce TCB

- SGX protects a **small** codebase
  - good!
- Protected process = “**Enclave**”
- Provides a trusted environment:
  - app integrity
  - protects data
SGX leaks privacy in many ways
Hyperthreading, Speculation, Page Tables, Caches, ...
SGX Uses Enclaves for Attestation (EPID)

Software uses attestation key to sign results of computation

Cache timing attacks could leak the key

Foreshadow, Usenix Security
Sanctum Secure Processor
No Speculation, No Hyperthreading

Partitioned
Last Level Cache
Sanctum’s Chain of Trust

- Manufacturers
- Certificate Authority
- Sanctum HW
- Security Monitor (SM)
- OS
- Process

Privileged (supervisor mode)
Unprivileged (user mode)

Strongly Isolated Enclave
Isolated Page Tables

Enclave A Virtual Address Space

- Host application space
- EVRANGE A

Enclave B Virtual Address Space

- Host application space
- EVRANGE B

Physical Memory

- OS region
- OS page tables
- Enclave A region
- Enclave A page tables
- Enclave B region
- Enclave B page tables
- OS region
- Enclave A region
Sanctum Secure Processor
No Speculation, No Hyperthreading

RISCV Rocket Core, Changes required by Sanctum (+ ~2% of core)

Also requires 9 new config registers
Status

• Sanctum on AWS F1—you too can use it (or break it!)

• *Ongoing*: Keystone processor on HiFive Unleashed RISC-V chip (with Krste Asanovic and Dawn Song, UCB)

• *Near future*: Out-of-order “Sanctoom” processor

• *Near future*: Formal verification effort (with Adam Chlipala, MIT)
In Conclusion,

• Significant security concerns with outsourcing computation especially to public clouds

• Intel’s SGX helps but leaks privacy through software side channels and is quite opaque

• Rethinking processor architecture to not sacrifice isolation and privacy when optimizing for performance
Thank you!