Foreshadow:
speculative attacks on SGX and beyond

Mark Silberstein

Joint work with
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx
Big picture in one slide

- Where do CPUs loose performance?
  - Branches, Memory translation
  - Technology scaling does not help
Big picture in one slide

• Where do CPUs lose performance?
  – Branches, Memory translation
  – Technology scaling does not help

• Speculative execution for latency hiding
  – CPU speculates the outcome of slow operations
  – **Continues** execution assuming speculation is correct
  – **Rolls back** the modified architectural state otherwise
Speculative execution attacks exploit

- Speculation past Illegal memory accesses
- Inability to entirely roll back μarch state
Today

- Background
- From Meltdown to Foreshadow
- SGX: Collateral damage
- Foreshadow-NG (L1TF)
- Discussion
Speculative execution 101

Instruction stream

1
2
3
4
5
6

Completed instructions

CPU

Execute

Retire (commit results)

Slow instruction

Depend on 3
Speculative execution 101

Instruction stream

Completed instructions

CPU

Execute

Retire (commit results)
Speculative execution 101

Instruction stream

Completed instructions

CPU

1. Retire (commit results)
2. Execute

3
4
5
6
Speculative execution 101

Instruction stream

Slow

Completed instructions

T_{slow}

1

Execute

2

Retire (commit results)

4

5

6
Speculative execution 101

Instruction stream

Slow

Execute

Retire

Executed speculatively

T_{slow}

1

2

5

6

Completed instructions
Speculative execution 101

Instruction stream

CPU

T_{slow}

Execute

Retire

Completed instructions

Not committed yet!
Speculative execution 101

Instruction stream

Completed instructions

Only 2 \textit{transiently} executed instructions fit in the speculation window

October 2018  
Mark Silberstein, Technion
Speculative execution 101

Instruction stream

Completed, speculation was right

Execute

Retire

Completed instructions

1

2

6

5

4
Speculative execution 101

Instruction stream

CPU

Completed instructions

Execute

Retire

Commit all pending instructions
Speculative execution 101

Instruction stream

Completed instructions

Execute

Retire

Completed, speculation was wrong

1

2

3

4

5

6
Speculative execution 101

Instruction stream

Completed instructions

CPU

Execute

Retire

Roll back architectural speculative state and continue execution
Prerequisites to speculative execution attack

- CPU speculates *insecurely*
- Speculative state **cannot be** roll back
- Race condition: roll back vs. leaking logic
  - Attack succeeds only if $T_{\text{speculative}} < T_{\text{slow}}$
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

Transient instructions

Access generates exception

3

movb (secret), %al

4

test %al, 0x1

5

je 1f

6

jmp 2f

1: leak(%al)

2: nop
Complete example (Rogue cache read – aka Meltdown)

Instruction stream

```
test %al, 0x1
je 1f
jmp 2f
1: leak (%al)
2: nop
```

movb (secret),%al

Execute

Retire

Slow: access to invalid (not present/not readable) address triggers exception that requires long time to resolve
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

```assembly
je 1f
jmp 2f
1: leak (%al)
2: nop
```

Execute

```assembly
movb (secret),%al
```

Retire

```assembly
test %al, 0x1
```

(secret) is *insecurely speculated* read from cache (or even DRAM)
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

Execute

movb (secret),%al

leak (%al)

Retire

test %al, 0x1

je 1f

Here the secret gets leaked
Execute

Rogue cache read – aka Meltdown

Instruction stream

test %al, 0x1
je 1f
jmp 2f
1: leak (%al)
2: nop

movb (secret), %al

Exception

Speculative state is cleaned except for the one leaked
Recipe: Speculative read attacks

- **P**rovoke insecure speculation
- **W**in the race
- **N**otify the attacker
Question 1: where does insecure speculation occur?

- Meltdown: access to a page with Supervisor bit
- Spectre V1: mis-speculated branch
- Foreshadow/L1TF: access to a non-present page, or via an incorrect mapping

The data is speculatively fetched from cache/memory violating protection guarantees (OS/program)
Question 2: How to avoid rollback?

- Not all μarch state can be rolled back
- μarch state becomes architecturally visible!
  - Caches
  - Branch predictors
  - Performance counters
  - Contention on shared resources
- Simplest: cache covert channel (Metldown/Spectre)
Flush-Reload covert channel

- Flush the cache before the attack
- Allocate
  
  ```
  char leak_array[4096*256]
  ```
- Sender: leak() function
  
  ```
  leak_array[4096*secret]=1
  ```
- Receiver: probe the array to identify cached values
Question 3: How to win the race condition

- Make sure `leak_array` is in TLB
- Make sure access to secrets is fast (e.g., in cache)
- Try many times
  - Need ways to suppress the exception
- Unsuccessful attempts are zero-biased
Question 3: How to win the race condition

- Make sure `leak_array` is in TLB
- Make sure access to secrets is fast (e.g., in cache)
- Try many times
  - Need ways to suppress the exception
- Unsuccessful attempts are zero-biased

Plus some secret sauce that nobody really understands why works
Agenda

- Foreshadow
- Collateral damage on SGX
- Foreshadow-NG /L1TF
- Discussion
SGX: background

- Enclave: reversed sandbox
- Private code & data
  - Confidentiality
  - Integrity
  - Freshness
- Defends against privileged SW!
- HW acceleration
- Scales with CPU scaling
SGX memory
DRAM encrypted, cache in plain text

secret_foo():
...
*p = 1;

Enclave

Enclave Page Cache (EPC)
Address translation in enclaves

secret_foo():
...
*p = 1;

Enclave

Hardware
Address translation

System memory

Page table

OS

EPC
Accessing EPC from outside: abort page semantics

foo():
...
printf(*p) ;

Hardware Address translation

Page table

OS

System memory

EPC

Process
Accessing EPC from outside: abort page semantics

foo():
...
printf(*p);
Challenges of SGX attacks

- P – Abort page behavior suppresses exception
- W – Secrets must be in L1 cache
- N – Same as Meltdown
Understanding memory translation with SGX

Virtual to physical

Enclave Mode?

In EPC

Is mapping valid

In SGX physical

PF  Read EPC  Read FF  Regular
Overriding abort page semantics

- Idea 1: modify EPC page as non-present
  - user calls `mprotect(epc_mem, PROT_NONE)`
- Access to the page triggers exception. Speculative path reads `epc_mem` from L1 despite being in EPC
Overriding abort page semantics

• Idea 1: modify EPC page as non-present
  – user calls `mprotect(epc_mem, PROT_NONE)`

• Access to the page triggers exception. Speculative path `reads epc_mem from L1` despite being in EPC

• Why does it work?
  – Speculative path ignores SGX memory checks
Overriding abort semantics

- Idea 1: modify EPC page as non-present
  - *user* calls `mprotect(epc_mem, PROT_NONE)`
- Access to the page triggers exception.
  Speculative path **reads** `epc_mem` from L1 despite being in EPC

- Why does it work?
  - Speculative path ignores SGX memory checks

This attack works from user space!
Overriding abort page semantics

- Idea 2: access maliciously mapped page
  - kernel maps an EPC page into another enclave
Overriding abort page semantics

- Idea 2: access maliciously mapped page
  - kernel maps an EPC page into another enclave

- Why does it work?
  - Speculative path ignores inter-enclave protection checks
W

How to ensure secrets are in L1

- Single-stepping an enclave with SGX-Step
- Controlled side channel attack
- *Dumping enclave's memory without enclave execution*
Copy enclave's data into cache

- Enclave's VM managed by untrusted OS
- SGX features special instructions to swap-in/swap-out a page from EPC
  - ewb  - evict a page from enclave
  - eldu  - load a page into enclave
- eldu decrypts the page and keeps the outcome in L1
W

Keep enclave's data in L1 cache

• Sources of cache pollution
  – system calls
  – enclave exits
  – system noise
W
Keep enclave's data in L1 cache

• Sources of cache pollution
  – system calls => avoid mprotect in retries
  – enclave exits => suppress exceptions
  – system noise => isolate cores

As in meltdown
Summary so far

- SGX is vulnerable to speculative execution attacks
- Enclave's data in L1 can be accessed in a speculation due to exception
- L1 can be populated by enclave pages without executing the enclave
- Result: dump all enclave memory
Collateral damage: attacking SGX attestation
Remote attestation

• Essential for SGX ecosystem
• Enables a party trusting Intel to trust an enclave executed on a remote machine
Remote attestation

• Example: Netflix video player runs on your computer, receives secrets from Netflix.

• Remote attestation proves to Netflix that
  – The player is running on genuine Intel's hardware
  – The player's binary is a genuine one

Sponsored add:
An excellent primer on SGX 2.0 attestation: first talk at
http://cyber.technion.ac.il/2018-summer-school-on-cyber-computer-security
SGX Architectural Enclaves

• Implement remote attestation in software
• **Rely on SGX security guarantees**
• Trusted by Intel
Remote attestation
Offline: generating EPID signing key

EPID=enhanced privacy ID
Remote attestation
Offline: generating EPID signing key

EPID=enhanced privacy ID

Provisioning Enclave

Generate Platform-specific EPID key

Use seal key to encrypt EPID key

Intel Provisioning Service
Remote attestation
Online: validation

Application enclave

Remote Verifier
Remote attestation
Online: validation
Remote attestation
Online: validation

Quoting enclave

Application enclave

Remote Verifier

EPID key
Remote attestation
Online: validation
Remote attestation
Online: validation

Quoting enclave
EPID key
Application enclave
Remote Verifier
Intel Quoting Service
Observations

- Knowing EPID key allows signing faked enclaves
- EPID is designed for unlinkability
  - Intel cannot tell apart enclaves signed with the same key
- Corollary: enclave user (Netflix client) cannot tell apart genuine and faked enclaves!
How to retrieve EPID?

- EPID is stored on a disk encrypted with sealing key
- Sealing key is found in enclave memory of the Quoting Enclave

We attack the Quoting Enclave:
A combination of
1. Controlled side channel
2. Foreshadow for extracting registers
AaaS
(Attestation as a Service)

- **@ForeshadowAaaS**
  Will attest to anything tweeted at it
- Reduced cost of hackering – no need to buy an SGX machine
- Hacker’s privacy guaranteed by EPID protocol
- Attestation server returns Group_Out_Of_Date
- SGX Keys are still not revoked (despite weeks of advances notice)
- Blocked by Twitter
Summary so far

- SGX is vulnerable to speculative execution attacks
- Allows dumping enclave's memory
- Attack enables leaking sealing key and EPID from infrastructural enclaves
- Breaks the SGX remote attestation without way to revoke (anonymous) EPID
Foreshadow-NG: L1TF

- Foreshadow reported on Jan 3\textsuperscript{rd} by KU Leuven, Jan 23\textsuperscript{rd} by Technion/Michigan/Adelaide
- Intel's follow up (Aug 11, but known since March): there are three other flavors, same bug

- Process-to-process
- Process-to-SMM
- VM guest to host
L1 Terminal Fault

• When an accessed page is marked \textit{not present (terminal fault)}, PA is used to access L1 cache, while ignoring:
  – SGX: EPC access checks
  – OS: Protection checks
  – VirtualMachine: GuestP-to-HostP translation

• Implication: guest controls which HostP addresses to access

• Major issue: forced 8 months of disclosure embargo
Foreshadow vs. Meltdown

- Spectre/Meltdown – same address space leaks
- Foreshadow – both intra and inter-address space leaks. Memory isolation non-existent
Mitigation: Foreshadow

- SGX microcode updates
  - flush L1 on each enclave exit/eldu
    => prevents non-concurrent attacks on L1
  - hyperthreading is part of the enclave trusted state
    => prevents concurrent attacks on L1
  - increase security version (TCB update)
Mitigation: L1TF

• Hypervisor:
  – no co-location of untrusted VMs on hyperthreads
  – New L1 flush instruction
  – Zero out non-present EPT entries
  – Dummy page at offset 0 in hypervisor
Open questions

- Foreshadow: bug or design flaw?
- Does SGX *inherit* the bug?
- What do we actually know about the reasons?
  - Hint: not much

- Disclosure process: who is in charge for the world piece?
Summary: Foreshadow

- PWN on enclaves
- Breaks SGX confidentiality
- Steals seal-key – breaks the integrity of persistent storage
- Breaks the remote attestation guarantees which relies on the enclave
- Same bug causes VM, OS and SMM protection violation
Questions?

mark@ee.technion.ac.il