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 fully roll back μarch state
- Covert/side channels to leak the 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

Slow instruction

Depend on 3

Completed instructions

CPU

Execute

Retire (commit results)
Speculative execution 101

Instruction stream

1. Execute
2. 3
3. 4
4. 5
5. 6

Completed instructions

1. Retire (commit results)

CPU
Speculative execution 101

Instruction stream

Completed instructions

CPU

2. Execute

1. Retire (commit results)
Speculative execution 101

Instruction stream

Slow

T_{\text{slow}}

Execute

Retire (commit results)

Completed instructions
Speculative execution 101

Instruction stream

Completed instructions

Slow

T_{\text{slow}}

Execute

Retire

Executed speculatively

1

2

3

4

5

6
Speculative execution 101

Instruction stream

CPU

Completed instructions

T_{\text{slow}}

Not committed yet!

Execute

Retire

1

2

3

5

4

6
Speculative execution 101

Instruction stream

Completed instructions

CPU

T_{slow}

Execute

Retire

T_{speculative}

Only 2 *transiently* executed instructions fit in the speculation window
Speculative execution 101

Instruction stream

- Execute
- Completed, speculation was right
- Completed instructions

- Retire
- Completed instructions

- 1
- 2
- 3
- 4
- 5
- 6
Speculative execution 101

Instruction stream

CPU

Completed instructions

1

2

3

4

5

Commit all pending instructions

Execute

Retire

6
Speculative execution 101

Instruction stream

Completed instructions

Completed, speculation was wrong

Execute

Retire

5

4

3

1

2

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** rolled back: data leak
- Race condition: roll back vs. leaking logic
  - Attack succeeds only if $T_{\text{speculative}} < T_{\text{slow access}}$
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

Transient instructions

Access generates exception

movb (kernel secret), %al

leak(%al)
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

slow: illegal access to an inaccessible address triggers *exception* that requires long time to resolve

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

leak (%al)

Execute

Retire
Complete example (Rogue cache read – aka Meltdown)

Instruction stream

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

Execute

```
leak(%al)
```

Retire

(secret) is **insecurely speculated**: read from cache or DRAM ignoring page protection
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

movb (secret),%al

Execute

leak(%al)

Retire

Need to be fast to finish before the exception is resolved
Complete example
(Rogue cache read – aka Meltdown)

Instruction stream

Exception

movb (secret),%al

Execute

Retire

leak (%al)

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

- Proveke insecure speculation
- W in the race
- Notify the attacker
Question 1: where does insecure speculation occur?

- Meltdown: exception due to access to a page with Supervisor bit
Question 1: where does insecure speculation occur?

- Meltdown: exception due to access to a page with Supervisor bit
- Spectre V1: mis-speculated branch
Question 1: where does insecure speculation occur?

- Meltdown: *exception* due to access to a page with Supervisor *bit*
- Spectre V1: mis-speculated branch
- Foreshadow/L1TF: *exception* due to access to a *non-present* page, or via an *incorrect* mapping
Question 1: where does insecure speculation occur?

- Meltdown: exception due to access to a page with Supervisor bit
- Spectre V1: mis-speculated branch
- Foreshadow/L1TF: exception due to 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 misspeculation 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
- Sender/receiver: declare
  
  ```c
  char leak_array[4K*256]
  ```

- Sender:
  
  ```c
  void leak_byte(char secret) {
    leak_array[4K*secret]=1;
  }
  ```

- Receiver: probe the array to identify cached values
  
  - `argmin(access_time(leak_array[4K*i]))`
Question 3: How to win the leak-to-rollback race condition

- Access to `leak_array` must be fast (in TLB)
- Access to secrets must be fast (in cache)
- Try many times
  - suppress the exception bailout
- Unsuccessful attempts are zero-biased
Question 3: How to win the leak-to-rollback race condition

- Access to `leak_array` must be fast (in TLB)
- Access to secrets must be fast (in cache)
- Try many times
  - suppress the exception bailout
- Unsuccessful attempts are zero-biased

Plus some secret sauce that nobody really understands why it works
Agenda

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

- Enclave: reversed sandbox
- Private code & data
  - Confidentiality
  - Integrity
  - Freshness
- Defends against privileged SW!
- HW acceleration
- Scales with CPU scaling

---

Application

Enclave

Enclave

Operating system
Background: SGX memory
DRAM encrypted, cache in plain text

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

Enclave Page Cache (EPC)
Background:
Address translation in enclaves

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

Enclave

Hardware
Address translation

Page table

OS

System memory

EPC
Background: SGX abort page semantics

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

Hardware
Address translation

Page table

OS

System memory

EPC

Process
Background: SGX abort page semantics

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

read 0xFFF

Process

Page table

OS

System memory

Hardware Address translation

EPC
Foreshadow uses speculative execution to leak secrets from SGX secure memory (EPC)
Agenda

- Foreshadow
- Collateral damage on SGX
- Foreshadow-NG /L1TF
- Discussion
Reminder: Speculative read attacks

- **P**rovoke insecure speculation
- **W**in the race
- **N**otify the attacker
Challenges of SGX attacks

- **Provoke**
  - Abort page behavior suppresses exception: no speculation

- **Provoke/Win** – Secrets must be in L1 cache

- **Notify** – Same as Meltdown

SGX is resilient to strawman Meltdown attack
Challenges of SGX attacks

- **Provoke**
  - Abort page behavior suppresses exception: no speculation

- **Provoke/Win** – Secrets must be in L1 cache

- **Notify** – Same as Meltdown

SGX is resilient to strawman Meltdown attack
Understanding memory translation with SGX

Virtual to physical

Enclave Mode?

In EPC

Is mapping valid

Exception

Read EPC

In SGX physical

Read FF

Regular
Understanding memory translation with SGX

Virtual to physical

Enclave Mode?

no

In EPC

no

Is mapping valid

no

Exception

Read EPC

Abort page path

no

In SGX physical

no

Read FF

Regular
Understanding memory translation with SGX

**Dangerous speculation happens here**

1. Virtual to physical
2. Enclave Mode?
   - no
3. In EPC
   - no
4. Is mapping valid
   - no
5. Exception

6. In SGX physical
   - no
   - Read FF
   - Regular
Provoke 1
Overriding abort page semantics

• Idea 1: access to a “not-present” EPC page
  – user calls mprotect(epc_mem, PROT_NONE)
Provoke 1
Overriding abort page semantics

• Idea 1: access to a “not-present” EPC page
  – user calls mprotect(epc_mem, PROT_NONE)

• Access to the epc_mem triggers exception

• Speculative path reads epc_mem from L1 despite SGX protection
Provoke 1

Overriding abort page semantics

- Idea 1: access to a “not-present” EPC page
  - user calls `mprotect(epc_mem, PROT_NONE)`
- Access to the `epc_mem` triggers exception
- Speculative path reads `epc_mem` from L1 despite SGX protection

- Why does it work?
  Speculative path ignores SGX memory checks
Provoke 1
Overriding abort page semantics

• Idea 1: access to "not-present" EPC page
  – user calls `mprotect(epc_mem, PROT_NONE)`
• Access to the `epc_mem` triggers exception
• Speculative path reads `epc_mem` from L1 despite SGX protection

• Why does it work?
  Speculative path ignores SGX memory checks!

This attack works from user space!
Provoke 2
Overriding abort page semantics

• Idea 2: access maliciously mapped page
  – kernel maps an EPC page into another enclave
Provoke 2
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
Attack works with secrets in L1!
How to ensure they are in L1?

1. Single-stepping an enclave with SGX-Step
2. Controlled side channel attack
3. Dumping enclave's memory without enclave execution via enclave swapping
Leak secret

• Same as in Meltdown:
  – flush-and-reload cache covert channel

• Some tweaking to win the race
Summary so far

- SGX is vulnerable to speculative execution attacks
- Enclave's data in L1 cache can be accessed via speculative access
- L1 cache can be populated via enclave paging mechanism 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**
  - keep Intel-provisioned Secret in the Architectural Enclave
- Trusted by Intel
Observations

- Knowing **Intel Secret** allows signing faked enclaves
- **Intel Secret** is *designed for unlinkability*
  - Intel cannot tell apart enclaves signed with the same key
- Corollary: with the Intel Secret in attacker's hands, enclave *users* (Netflix) cannot tell apart genuine and faked enclaves!
How to retrieve Intel Secret?

- The Secret is stored on a disk encrypted with sealing key
- Sealing key is found in enclave's memory of the Intel Architectural Enclave
How to retrieve Intel Secret?

- The Secret is stored on a disk encrypted with sealing key
- Sealing key is found in enclave's memory of the Intel Architectural Enclave

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

- @ForeshadowAaaS Will attest to anything tweeted at it
- Reduced cost of hackership – 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 Secret from infrastructural enclaves
- **Breaks the SGX remote attestation** without an easy way to revoke (anonymous) Secret
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 *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 Host Physical addresses to access

- Major issue: forced 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 (methodology) flaw?
• Does SGX *inherit* the bug from X86?
• What do we actually know about the reasons?
  – Hint: not much
• SGX remote attestation relies on SGX – poor design choice?
• Disclosure process: who is in charge for the world peace?
Summary: Foreshadow

- PWN SGX 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
Backup
Provoke/Win race
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
Provoke/Win race

Keep enclave's data in L1 cache

• Sources of cache pollution
  – system calls
  – enclave exits
  – system noise
Provoke/Win race
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
Remote attestation
Offline: generating EPID signing key

EPID=enhanced privacy ID
Remote attestation

Offline: generating EPID signing key

EPID=enhanced privacy ID

- Generate Platform-specific EPID key
- Use seal key to encrypt EPID key
Remote attestation
Online: validation
Remote attestation
Online: validation
Remote attestation
Online: validation

Quoting enclave

EPID key

Application enclave

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

Quoting enclave

Application enclave

Intel Quoting Service

Remote Verifier

EPID key