Speculative Execution Vulnerabilities: From a Simple Oversight to a Technological Nightmare

Raoul Strackx
raoul.strackx@cs.kuleuven.be
@raoul_strackx
imec-DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001 Belgium

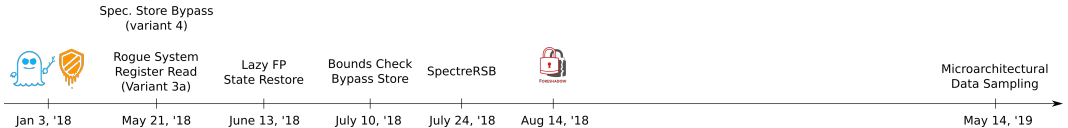
Hardwear.io, June 14th, 2019

DistriNet

2018 started very terrifying/exciting...

- **Spectre**: Extract data from running processes
- **Meltdown**: Read *full* RAM contents

. . . and continued along the same path



Comparing Foreshadow/Meltdown/Spectre/...

	Title	CVE	SA	Severity	Disclosure Date
6.5	Microarchitectural Data Sampling	CVE-2018-12126, CVE-2018-12127, CVE-2018-12130, CVE-2019-11091	INTEL-SA-00233	Medium	2019-05-14
7.3	L1 Terminal Fault	CVE-2018-3615, CVE-2018-3620, CVE-2018-3646	INTEL-SA-00161	High	2018-08-14
4.3	Rogue System Register Read	CVE-2018-3640	INTEL-SA-00115	Medium	2018-05-21
4.3	Speculative Store Bypass	CVE-2018-3639	INTEL-SA-00115	Medium	2018-05-21
5.6	Branch Target Injection	CVE-2017-5715	INTEL-SA-00088	Medium	2018-01-03
5.6	Bounds Check Bypass	CVE-2017-5753	INTEL-SA-00088	Medium	2018-01-03
5.6	Rogue Data Cache Load	CVE-2017-5754	INTEL-SA-00088	Medium	2018-01-03

Figure: source: `https://software.intel.com/security-software-guidance/software-guidance`

Foreshadow Attacks

- Independently discovered
- Team of KU Leuven, Belgium
- Team of Universities of Technion, Michigan and Adelaide and DATA61
- Intel discovered other variants

`foreshadowattack.eu`

KU LEUVEN Technion Israel Institute of Technology UNIVERSITY OF MICHIGAN THE UNIVERSITY of ADELAIDE DATA 61

Foreshadow Attacks

- Independently discovered
- Team of KU Leuven, Belgium
- Team of Universities of Technion, Michigan and Adelaide and DATA61
- Intel discovered other variants

`foreshadowattack.eu`

FORESHADOW: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution

Jo Van Bulck[1], Marina Minkin[2], Ofir Weisse[3], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Thomas F. Wenisch[3], Yuval Yarom[4], and Raoul Strackx[1]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

Abstract

Trusted execution environments, and particularly the Soft- distrusting enclaves with a minimal Trusted Computing Base (TCB) that includes only the processor package and microcode. Enclave-private CPU and memory state is

Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution
Revision 1.0 (August 14, 2018)

Ofir Weisse[1], Jo Van Bulck[1], Marina Minkin[2], Daniel Genkin[3], Baris Kasikci[3], Frank Piessens[1], Mark Silberstein[2], Raoul Strackx[1], Thomas F. Wenisch[3], and Yuval Yarom[4]

[1]*imec-DistriNet, KU Leuven,* [2]*Technion,* [3]*University of Michigan,* [4]*University of Adelaide and Data61*

Abstract

In January 2018, we discovered the *Foreshadow* transient execution attack (USENIX Security'18) targeting Intel tion requires different computational tasks belonging to separate security domains to be isolated from each other and prevented from reading each other's memory. In modern computer architectures this is typically achieved

KU LEUVEN Technion Israel Institute of Technology UNIVERSITY OF MICHIGAN THE UNIVERSITY of ADELAIDE DATA 61

These were vulnerabilities in the processor itself
Hence, virtually *every* application was effected!

This led to various reactions

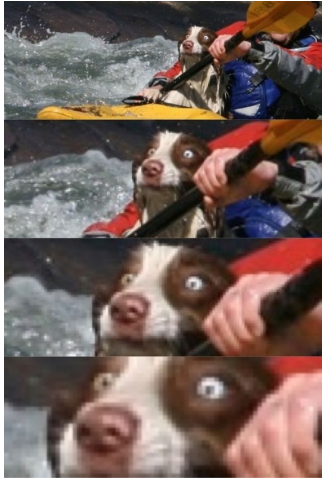
How we told our upper management at the university (Nov '17)...

Figure: source: `https://pin.it/k4j53t23xiiqcd`

How we told Intel (Jan ’18). . .

Figure: source: `https://pin.it/k4j53t23xiiqcd`

How IT professionals reacted (to this class of vulnerabilities)...



How Intel stock owners reacted. . .

How do these attacks work, in general?

. . . Side-channel attacks

Figure: The Italian Job (source: `imdb.com`)

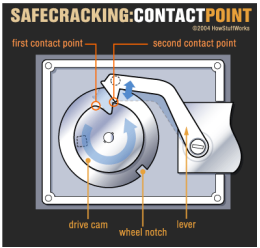
Attacker

Charlize Theron

action: rotate & listen

carrier: sound

Victim



Vault

Security flaw: Lever may produce sound

sources: `https://home.howstuffworks.com/`, `imdb.com`

How does the Foreshadow attack work?

One vulnerability to rule them all

- **Foreshadow-OS**: Bare-metal not-present pages
- **Foreshadow-VMM**: VM guest page tables
- **Foreshadow-SGX**: Intel SGX enclaves
- **Foreshadow-SMM**: Attacking System Management Mode

→ The target heavily affects how the attack can be launched

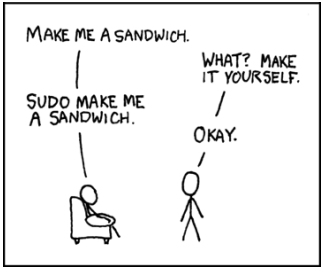
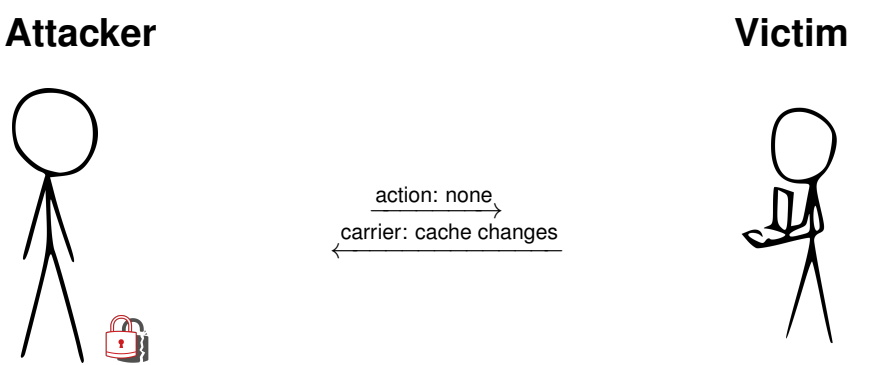


Figure: source: xkcd.com/149/

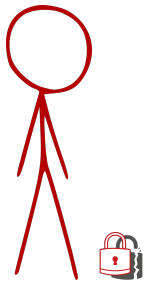
Luckily, these attacks can “only” read privileged memory

Foreshadow-OS: Reading L1 data through *bare-metal* not-present pages...

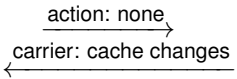


Security flaw: OoO execution leaves traces of transient instructions

Attacker



Victim



Foreshadow-OS

Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

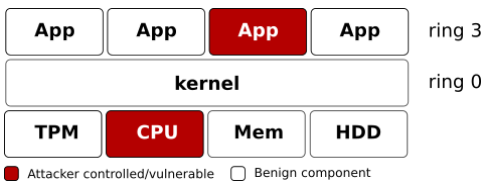
Setting: Attacker-controlled process

Attack model:

- Attacker operates within a malicious process
- Benign, bare-metal kernel ensures process isolation

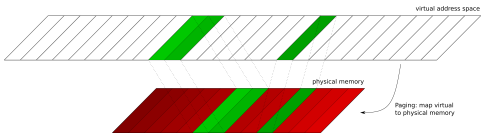
Attack objective:

- Read data outside the process' address space



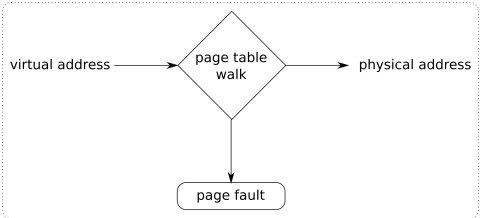
Background: How does process isolation work. . .

- MMU: map virtual address space to physical memory
- Protect physical memory by:
  - Not providing a mapping
  - Restricting access (e.g., U/S-bit)



Background: How does process isolation work. . .

- MMU: map virtual address space to physical memory
- Protect physical memory by:
  - Not providing a mapping
  - Restricting access (e.g., U/S-bit)



Background: How does process isolation work. . .

Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Figure: source: Intel 64 and IA-32 architectures software developer's manual

Background: How does process isolation work. . .

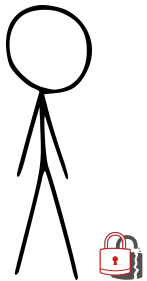
Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Figure: source: Intel 64 and IA-32 architectures software developer's manual

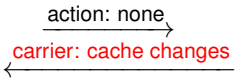
Background: How does process isolation work. . .

When P-bit is 0, the entry's physical address field may be re-used to keep track of the swapped out page

Attacker



Victim



Foreshadow-OS

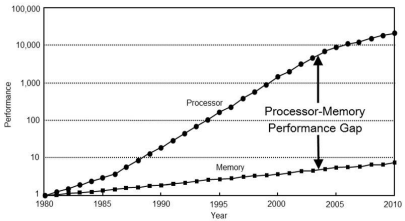
Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

The message carrier: How does the cache work?

Caching

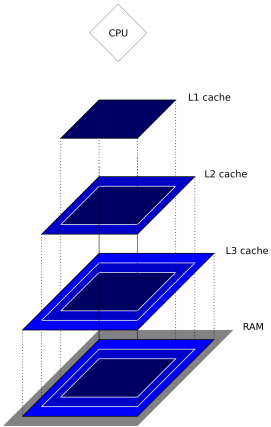
- Problem: Memory performance grows much slow than CPU performance
- Solution: fast but small caches
  - Intel 486: L1 cache (’89)
  - Intel Pentium Pro: L1 & L2 cache (’95)
  - Today: L1, L2 & L3 caches



The message carrier: How does the cache work?

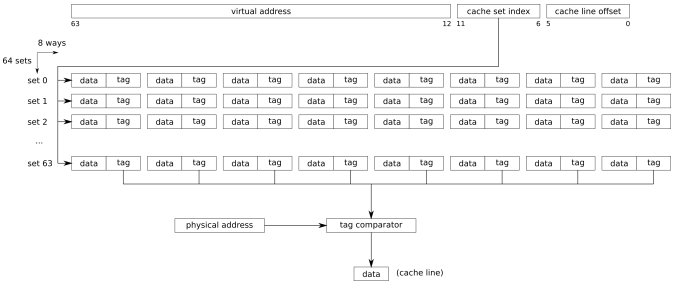
Caching

- Problem: Memory performance grows much slow than CPU performance
- Solution: fast but small caches
  - Intel 486: L1 cache (’89)
  - Intel Pentium Pro: L1 & L2 cache (’95)
  - Today: L1, L2 & L3 caches

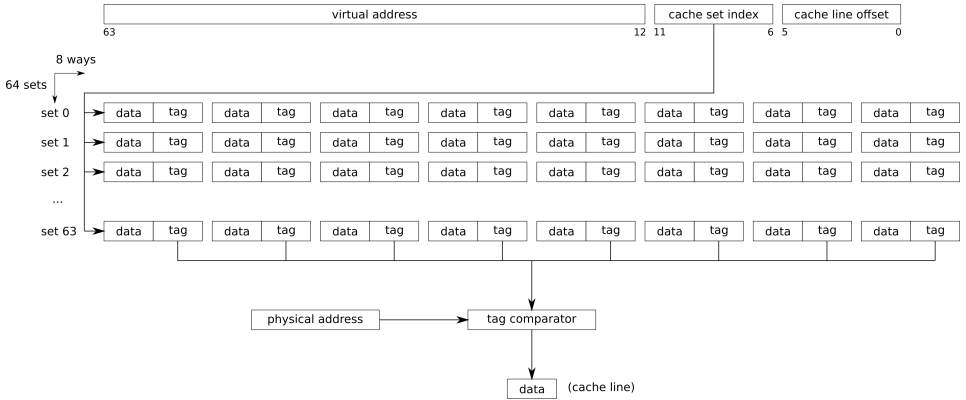


The message carrier: how does the cache work?

- Cache lines: 64 B
- L1:
  virtually-indexed,
  physically tagged
- 64 sets, 8 ways



The message carrier: how does the cache work?



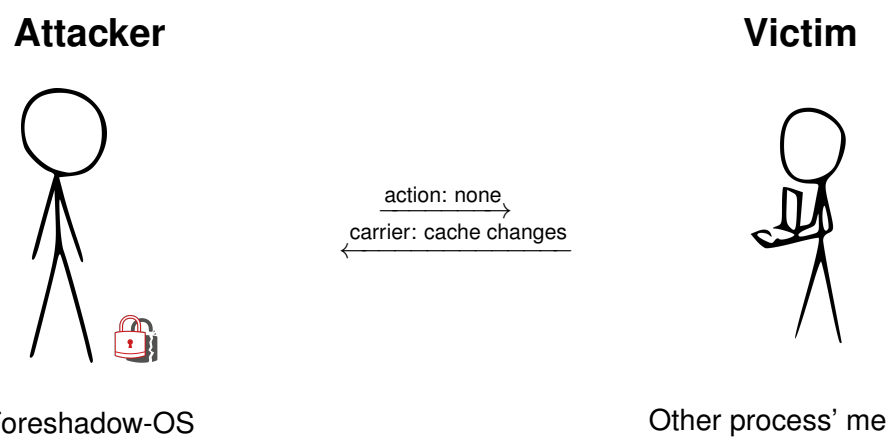
The message carrier: How does the cache work?

Manipulating the cache:

- Data accesses: load in L1-L3 cache
- `clflush`: Flush data from caches

memory	timing (in cycles)	std. dev.
L1	46	1.25
L2	53	1.14
RAM	246	6.22

$\rightarrow$ Any timing results <146 cycles clearly hits the cache



Security flaw: OoO execution leaves traces of transient instructions

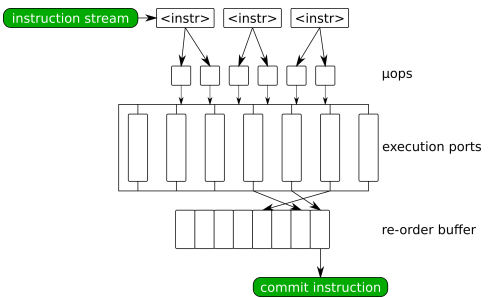
Background: Out of Order Execution

- **Problem**: We want more speed!
- **Solution**: Start executing instruction as soon as possible!
  - Pipeline instructions
  - Out-of-order execution of μops
  - (Speculative execution) $\rightsquigarrow$ see Spectre-like attacks

Background: Out of Order Execution

Out-of-order execution

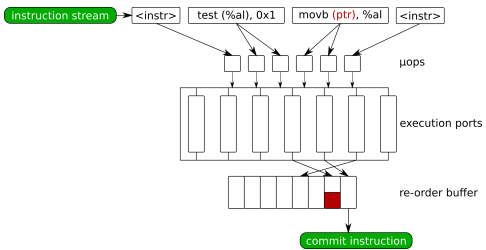
- Split instruction in μops
- Use multiple execution ports
- Execute μop as soon as possible
- Reorder ensures results/exceptions are visible in-order of instruction stream



The Security Flaw: Transient Execution

Transient execution:

- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started
- Processor rolls back architectural changes

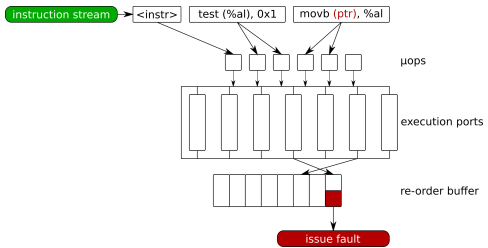


Key issue: Not all side-effects of “unreachable instructions” are rolled back correctly! (e.g., cache changes)

The Security Flaw: Transient Execution

Transient execution:

- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started
- Processor rolls back architectural changes

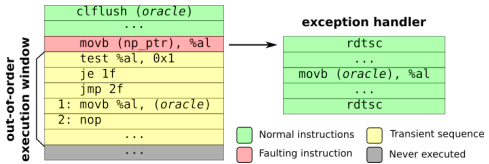


Key issue: Not all side-effects of “unreachable instructions” are rolled back correctly! (e.g., cache changes)

The Security Flaw: Transient Execution

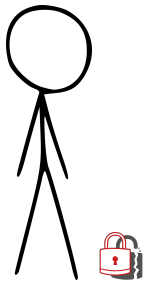
Transient execution:

- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started
- Processor rolls back architectural changes



Key issue: Not all side-effects of “unreachable instructions” are rolled back correctly! (e.g., cache changes)

Attacker



action: none

carrier: cache changes

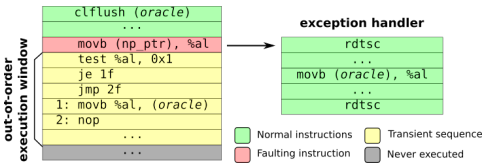
Victim

Foreshadow-OS

Other process' memory

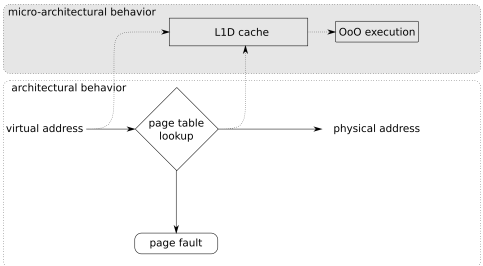
Security flaw: OoO execution leaves traces of transient instructions

Putting it all together



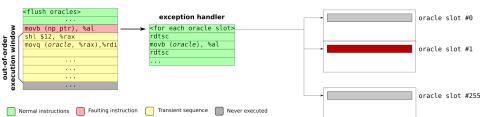

```
1 int8_t *oracle = ...;
2 int8_t *np_ptr = ...;
3
4 // Step 1: Remove variable oracle from cache
5 clflush( oracle );
6
7 // Step 2: Trick system in sensitive data in L1 but PTE present bit to 0
8
9 // Step 3: attempt to read not present memory
10 if ( *np_ptr == 1 )
11   // place oracle variable in the cache iff *np_ptr == 1
12   _tmp = *oracle;
13
14 // suppress fault
15
16 // Step 4: is oracle cached?
17 if ( time_access( oracle ) < 146 )
18   print( "sensitive data == 1!" );
19 else
20   print( "sensitive value != 1" );
```

Putting it all together



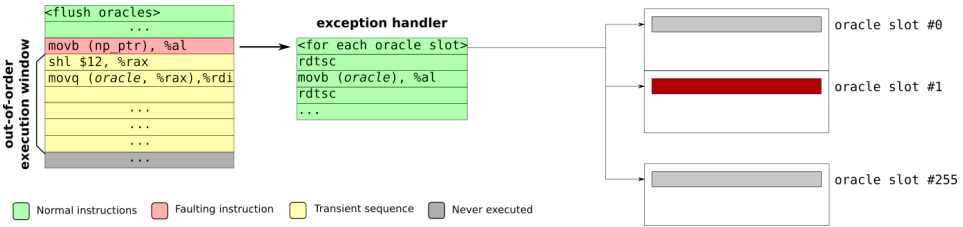

```
1 int8_t *oracle = ...;
2 int8_t *np_ptr = ...;
3
4 // Step 1: Remove variable oracle from cache
5 clflush( oracle );
6
7 // Step 2: Trick system in sensitive data in L1 but PTE present bit to 0
8
9 // Step 3: attempt to read not present memory
10 if ( *np_ptr == 1 )
11   // place oracle variable in the cache iff *np_ptr == 1
12   _tmp = *oracle;
13
14 // suppress fault
15
16 // Step 4: is oracle cached?
17 if ( time_access( oracle ) < 146 )
18   print( "sensitive data == 1!" );
19 else
20   print( "sensitive value != 1" );
```

Increasing the bandwidth of the attack




```
1 int8_t *oracles = ...;
2 int8_t *np_ptr = ...; // the secret
3 int8_t _tmp;
4
5 // Step 1: Remove oracle slots from cache
6 for ( int i = 0; i < 256; ++i )
7   clflush( &oracles[4096 * i] );
8
9 // Step 2: Trick system in sensitive data in L1 but PTE present bit to 0
10
11 // Step 3: attempt to read not present memory
12 _tmp = oracle[4096 * (*np_ptr)];
13
14 // suppress fault
15
16 // Step 4: which oracle slot is cached?
17 for ( int i = 0; i < 256; ++i ) {
18   if ( time_access( oracle[4096 * i] ) < 146 )
19     print( "*np_ptr = %i\n", i );
20 }
```

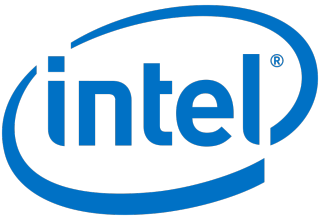
Increasing the bandwidth of the attack



Who's Affected?

Vulnerable processors:

- Intel Core processors of the last 7 years
- Intel server processors
- NOT AMD, not ARM

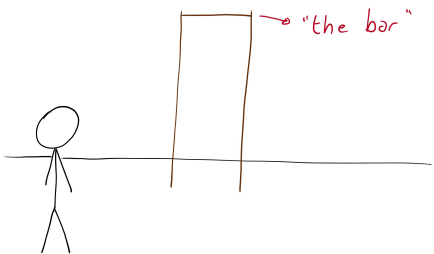


Impact of this attack

Requirements:

- Secret data in L1D
- Page must be not-present

⇝ Most difficult attack, “easiest” to understand
⇝ Low impact!

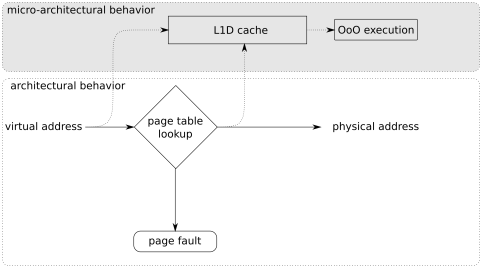


Mitigations

- Long term: Replace chips!
- Short term:
  - No readily apply-able microcode patch!
  - Software approaches:
    - Ensure PTE entry do not point to existing physical address
    - Use new instruction: `IA32_FLUSH_CMD` to flush L1D cache

Mitigations

- Long term: Replace chips!
- Short term:
  - No readily apply-able microcode patch!
  - Software approaches:
    - Ensure PTE entry do not point to existing physical address
    - Use new instruction: `IA32_FLUSH_CMD` to flush L1D cache



Mitigations

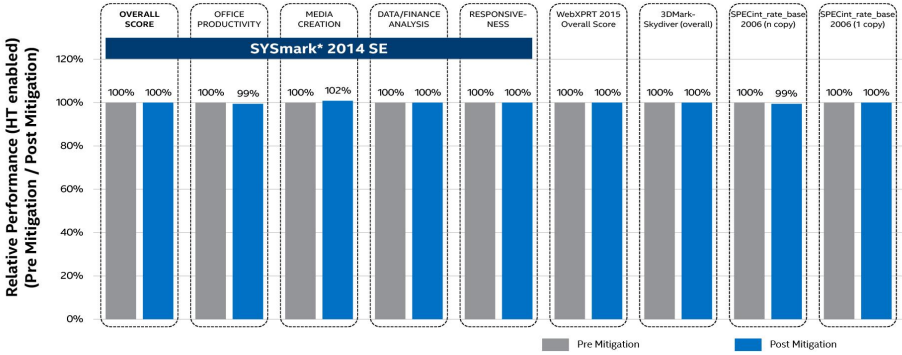
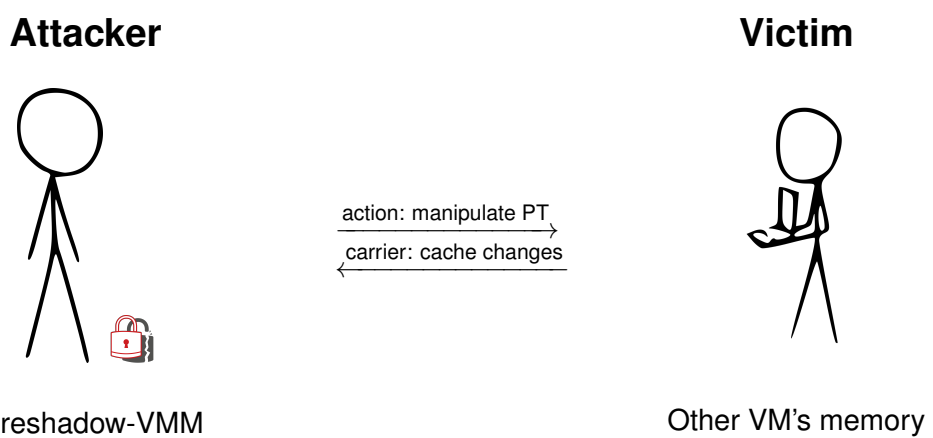


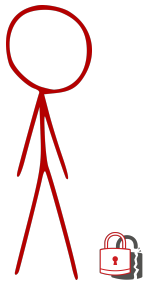
Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

Foreshadow-VMM: Reading physical L1 data through *virtualized* not-present pages. . .



Security flaw: OoO execution leaves traces of transient instructions

Attacker



Victim

action: manipulate PT

carrier: cache changes

Foreshadow-VMM

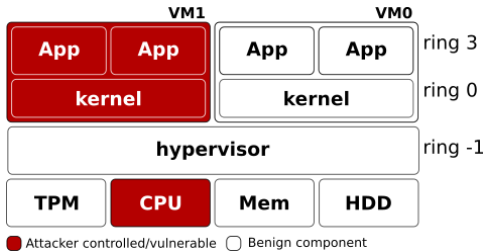
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Setting: Attacker-controlled VM

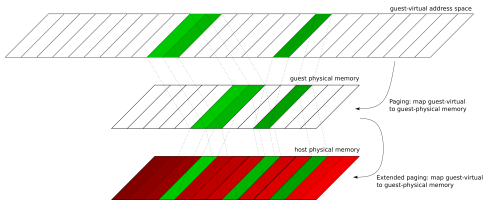
- Multiple VMs on one physical server
- Attacker-controlled VM
- Hypervisor ensures VM isolation

⇝ Goal: read other VMs data



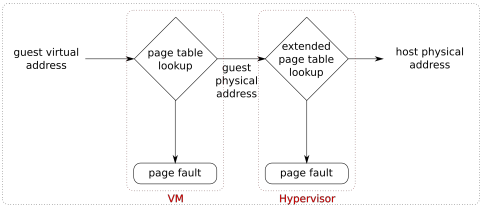
How do extended page tables work

- Adds another layer:
  - PT: guest-virtual address → guest-physical address
  - EPT: guest-physical address → host-physical address
- EPT: 4-level page table

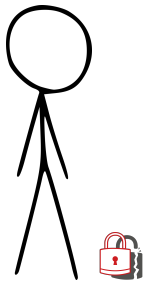


How do extended page tables work

- Adds another layer:
  - PT: guest-virtual address → guest-physical address
  - EPT: guest-physical address → host-physical address
- EPT: 4-level page table



Attacker



Foreshadow-VMM

action: manipulate PT

carrier: cache changes

Victim

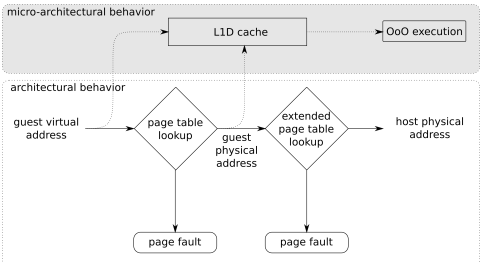
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

The Security Flaw: Interpreting guest-physical as host-physical addresses

- VM-level: non-present PTE entry
- VMM-level: irrelevant
- Upon access:
  - Tag data access as a violation
  - Pass *guest physical* address as *host physical* address to L1D cache
  - Continue transient execution!!

$\rightsquigarrow$ This breaks the VM's address space abstraction!



Foreshadow-VMM: The exploit

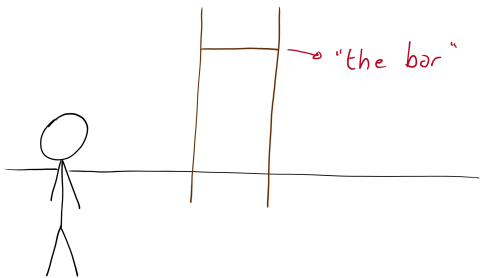

```
1  int8_t *oracles = ...;
2  int8_t *np_ptr = ...;
3  int8_t _tmp;
4
5  // Step 1: Setup PT to physical address of interest
6
7  // Step 2: Remove oracle slots from cache
8  for ( int i = 0; i < 256; ++i )
9    clflush( &oracles[4096 * i] );
10
11 // Step 3: Wait for sensitive data in L1D
12
13 // Step 4: attempt to read not present memory
14 _tmp = oracle[4096 * (*np_ptr)];
15
16 // suppress fault
17
18 // Step 5: is oracle cached?
19 for ( int i = 0; i < 256; ++i ) {
20   if ( time_access( oracle[4096 * i] ) < 146 )
21     print( "*np_ptr = %i\n", i );
22 }
```

Impact of Foreshadow-VMM

Requirements:

- Attacker must have full VM under her control
- Secret data must reside in L1D

⇝ Modest impact!

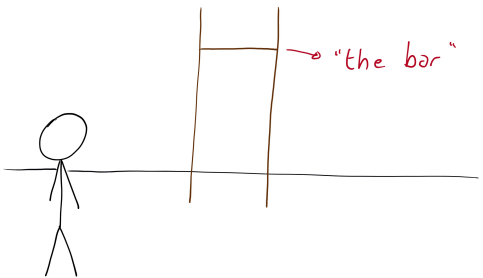


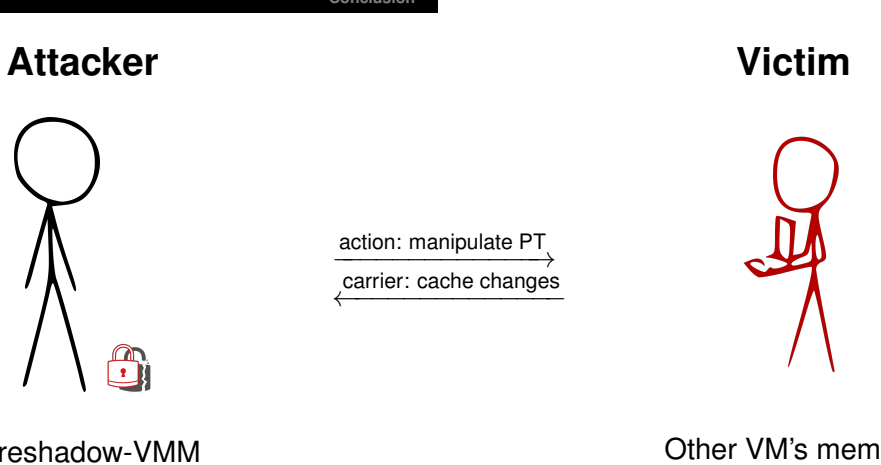
Impact of Foreshadow-VMM

Requirements:

- Attacker must have full VM under her control
- Secret data must reside in L1D ← **This may not be that complicated**

⇝ Modest impact!

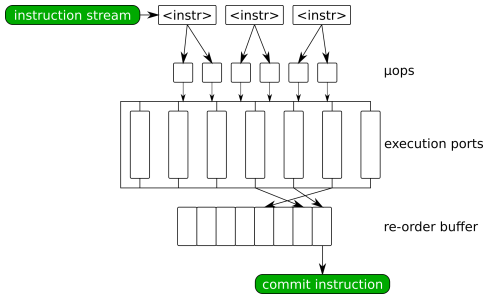




Security flaw: OoO execution leaves traces of transient instructions

Intel HyperThreading as an enabler

- Problem: Execution ports are still under-utilized
- Solution: Split physical core in two
- Duplicated HW:
  - register file
  - re-order buffer
  - ...
- Shared:
  - Execution ports
  - **L1 cache!** (and other levels)

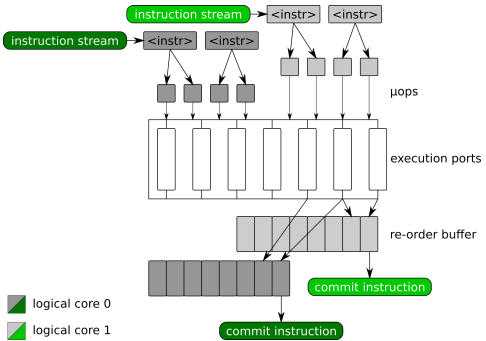


⇝ Performance increase of up to 30%[1]

https://www.cs.sfu.ca/~fedorova/Teaching/CMPT886/Spring2007/papers/hyper-threading.pdf

Intel HyperThreading as an enabler

- Problem: Execution ports are still under-utilized
- Solution: Split physical core in two
- Duplicated HW:
  - register file
  - re-order buffer
  - ...
- Shared:
  - Execution ports
  - **L1 cache!** (and other levels)



⇝ Performance increase of up to 30%[1]

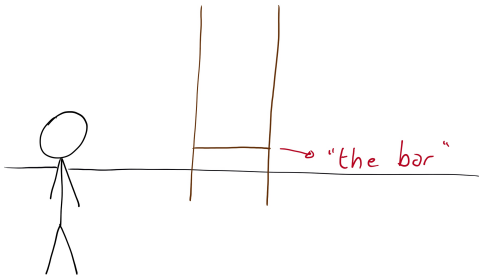
https://www.cs.sfu.ca/~fedorova/Teaching/CMPT886/Spring2007/papers/hyper-threading.pdf

Impact of Foreshadow-VMM (with HT enabled)

Requirements:

- Attacker must have full VM under her control
- Secret data must reside in L1D ← **Just have a little bit of patience!**

⇝ High impact!



Mitigations

Mitigations:

- Long term: Replace chips!
- Short term:
  - Make sure no secrets are in L1D cache

  → Flush L1D upon every VM-entry

  → Make sure no two different VMs execute on same physical core
    - Patch VM scheduler
    - Disable HyperThreading

Mitigations – Disabling HyperThreading

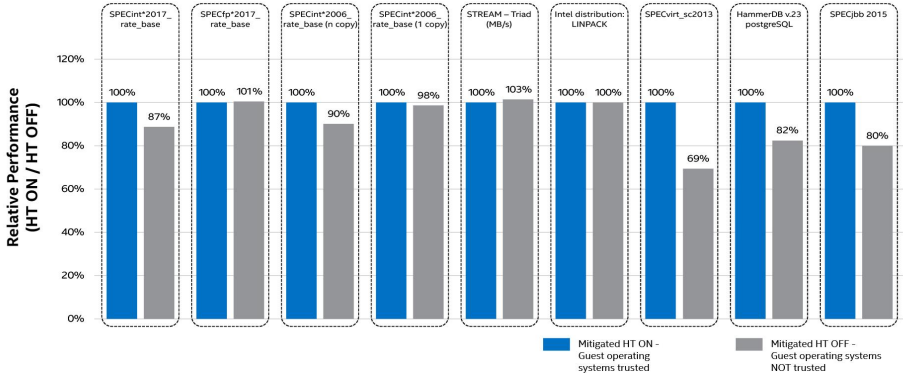


Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

Mitigations – Updating VM scheduler

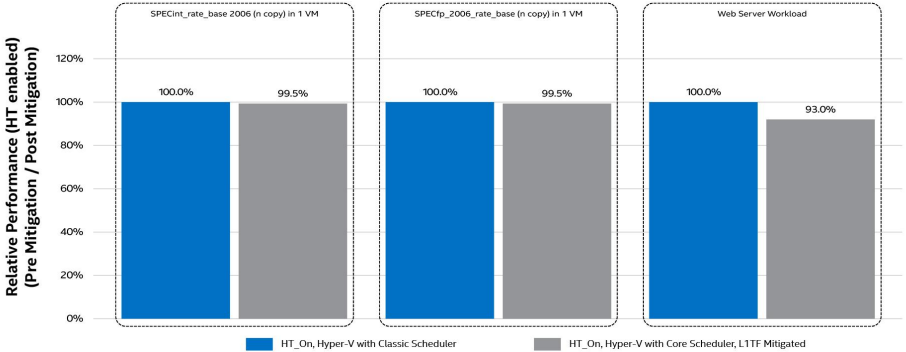
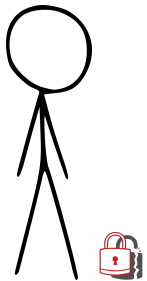


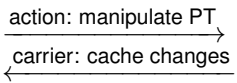
Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

Foreshadow-SGX: Dismantling Intel SGX's security objectives

Attacker



Victim



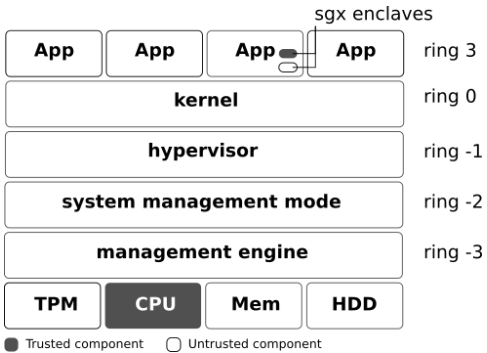
Foreshadow-SGX

SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

Background: Intel SGX

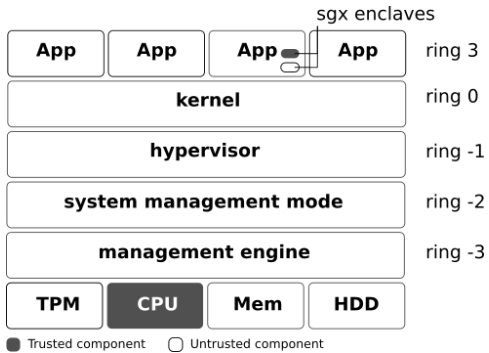
- **Problem**: Huge software TCB
- **Solution**: Protected-Module Architecture (e.g., Intel SGX)
- Only trust Intel hardware/enclaves
- Use cases:
  - protecting finger prints
  - DRM
  - Secure cloud-based processes
  - ...



Background: Intel SGX

Key properties:

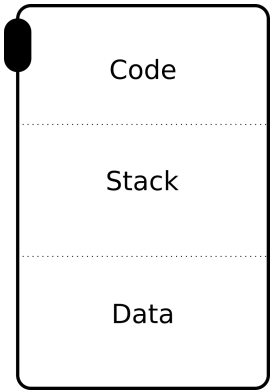
- Isolation
- Secure storage
- Attestation



Background: Intel SGX

Isolation:

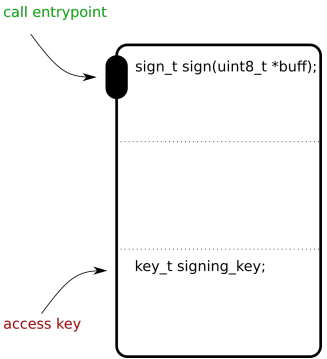
- Enclaves live in process' address space
- Only accessible through specific entry points
- Abort page semantics: Reading enclave memory outside the enclave results in `-1`.



Background: Intel SGX

Isolation:

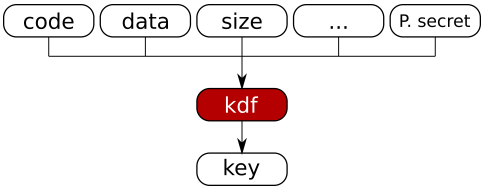
- Enclaves live in process' address space
- Only accessible through specific entry points
- Abort page semantics: Reading enclave memory outside the enclave results in `-1`.



Background: Intel SGX

Secure Storage:

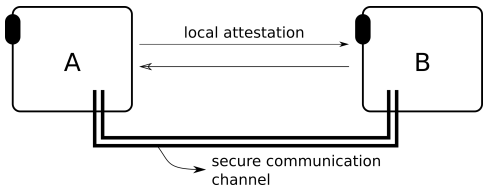
- Enclave die at loss of power
- Seal/Unseal confidential data
- Key derivation ensure unique key per enclave



Background: Intel SGX

Attestation:

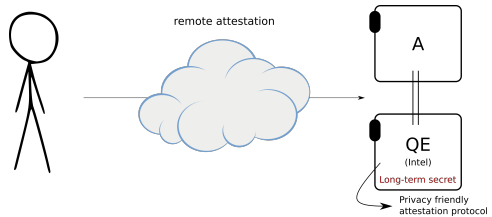
- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as building block for remote attestation
- EPID attestation protocol can ensure that attestation responses cannot be linked

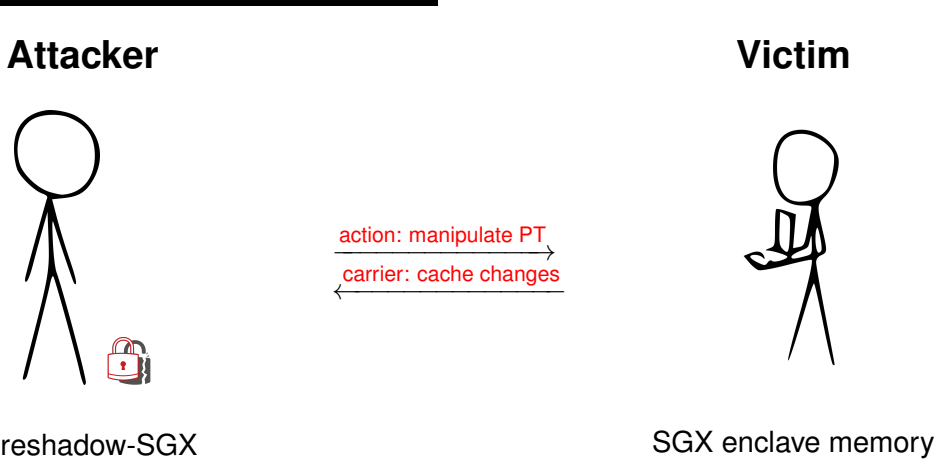


Background: Intel SGX

Attestation:

- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as building block for remote attestation
- EPID attestation protocol can ensure that attestation responses cannot be linked

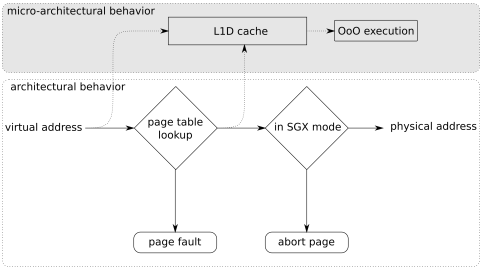




Security flaw: OoO execution leaves traces of transient instructions

The attack approach

- Bypass abort page semantics
- Ensure data in L1D:
  - Zero-step through enclave
  - Some instructions load enclave data in L1D as a side effect (e.g., `eldu`)



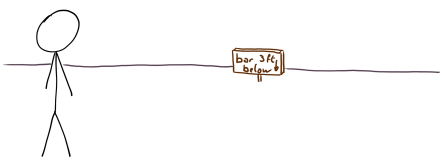
Impact of this attack

Requirements:

- Mark enclave page not-present
- Call enclave/issue `eldu` instruction

⇝ Completely breaks remote/local attestation, sealed storage, enclave isolation

⇝ Leaked Intel long-term SGX attestation keys



Mitigations

- Long term: Replace chips!
- Short term:
  - TCB recovery: increase CPU version number
  - Ensuring no secrets in L1 when enclave are not executing
  - Include status of HT during key derivation

Speculative Execution Attacks: Much ado about nothing?

No!

- Meltdown / Foreshadow-VMM/SGX are really powerful attacks

Yes (because we were lucky!)

- “Easiest” attacks, also easiest to mitigate
- Some (but very few) malware samples found abusing these exploits
- Mitigations were (roughly) in place at the time of disclosure

$\rightarrow$ I’m more worried about the next big speculative execution attack

Conclusion

- Foreshadow first transient execution attack that breaks the virtual memory abstraction, MDS are the second
- Speculative execution cannot be removed completely without a significant performance hit
- We have no idea how much leaky optimizations there are present in modern processors
- Modern x86 processors have become too complex to completely understand
- If possible, disable HyperThreading!

Thank you!

Thank you! Questions?

raoul.strackx@cs.kuleuven.be
@raoul_strackx