The Foreshadow Attack: 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

SecAppDev, February 22nd, 2019

DistriNet

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*

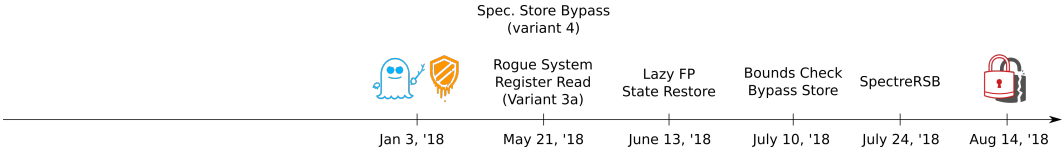
Abstract

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

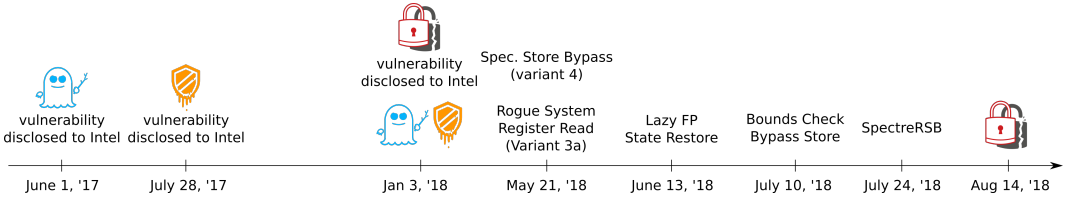
2018 started very terrifying/exciting. . .

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

. . . and continued along the same path



. . . and continued along the same path



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 my mother reacted. . .

How the media reacted (to this class of vulnerabilities)...

The New York Times

Researchers Discover Two Major Flaws in the World's Computers

Paul Kocher, left, moderating the RSA Conference 2016 in San Francisco. Mr. Kocher is an independent researcher who was an integral part of the team that discovered the flaws.
Jim Wilson/The New York Times

How the media reacted (to this class of vulnerabilities)...

THE WALL STREET JOURNAL.

Subscribe Now | Sign In

Home World U.S. Politics Economy Business Tech Markets Opinion Life & Arts Real Estate WSJ. Magazine Search

WeWork's CEO Makes Millions as Landlord to WeWork

Huawei Targeted in Criminal Probe for Alleged Theft of Trade Secrets

Tribune Publishing Recently Tried to Rekindle Merger Talks ...

'They Own the System': Amazon Rewrites Book Industry by ...

CIO JOURNAL.

The Morning Download: Spectre, Meltdown Response Demands 'No System Left Behind'

By Tom Loftus

Jan 4, 2018 8:23 am ET

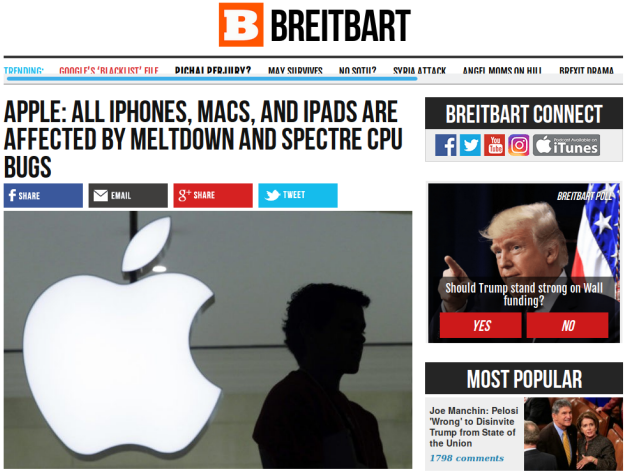
0 COMMENTS

In this July 20, 2011, photo, Intel Corp. offices are seen in Santa Clara, Calif. Intel says it is working to patch a security vulnerability in its products but says the average computer user will not experience significant slowdowns as the problem is fixed. PHOTO: ASSOCIATED PRESS

Recommended Videos

1. The Best and Worst U.S. Airlines of 2018
2. Breaking the Brexit Deadlock
3. PG&E's Bankruptcy Debacle: How Did We Get Here?
4. Barr Says He'll 'Not Be Bullied,' Defends Position on Mueller Probe
5. Theresa May's Brexit Plan Rejected by U.K. Parliament

How the media reacted (to this class of vulnerabilities)...



How the media reacted (to this class of vulnerabilities)...

Support The Guardian
Contribute → Subscribe →
Search jobs | Sign in | Search | The Guardian | International edition

News | Opinion | Sport | Culture | Lifestyle | More

World UK Science Cities Global development Football Tech Business Environment Obituaries

Data and computer security

Meltdown and Spectre: 'worst ever' CPU bugs affect virtually all computers

Everything from smartphones and PCs to cloud computing affected by major security flaw found in Intel and other processors - and fix could slow devices

Spectre and Meltdown processor security flaws - explained

Samuel Gibbs
Thu 4 Jan 2018 12.06 GMT

4,451

This article is over 1 year old

Meltdown and Spectre security flaws: so big they have their own logos. Photograph: tcareob72/Natascha Eibl/Getty Images/iStockphoto

most viewed

Live Brexit: May faces Corbyn at PMQs ahead of no-confidence vote - Politics live

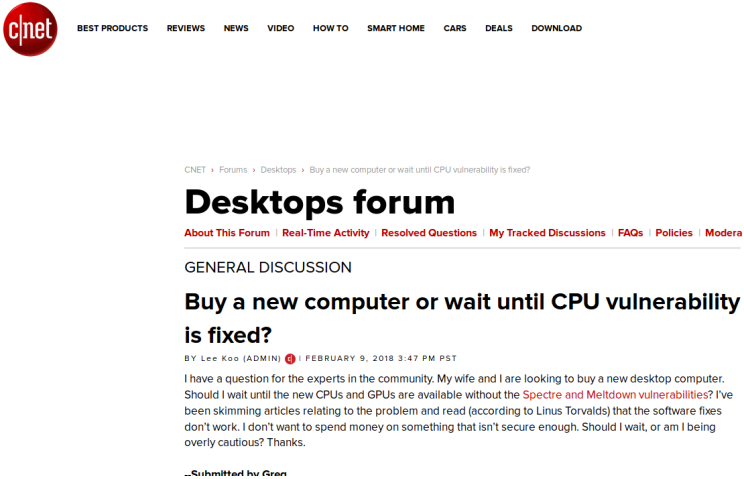
Theresa May suffers historic defeat in vote as Tories turn against her

Theresa May omits Jeremy Corbyn from cross-party Brexit talks

'You didn't get fired': Christie offers new evidence Trump avoids confrontation

Football transfer rumours: Arsenal to secure James Rodríguez for £3m?

How the general public reacted (to this class of vulnerabilities)...



cnet BEST PRODUCTS REVIEWS NEWS VIDEO HOW TO SMART HOME CARS DEALS DOWNLOAD

CNET › Forums › Desktops › Buy a new computer or wait until CPU vulnerability is fixed?

Desktops forum

About This Forum | Real-Time Activity | Resolved Questions | My Tracked Discussions | FAQs | Policies | Modera

GENERAL DISCUSSION

Buy a new computer or wait until CPU vulnerability is fixed?

BY Lee Koo (ADMIN) | FEBRUARY 9, 2018 3:47 PM PST

I have a question for the experts in the community. My wife and I are looking to buy a new desktop computer. Should I wait until the new CPUs and GPUs are available without the Spectre and Meltdown vulnerabilities? I've been skimming articles relating to the problem and read (according to Linus Torvalds) that the software fixes don't work. I don't want to spend money on something that isn't secure enough. Should I wait, or am I being overly cautious? Thanks.

--Submitted by Greg

. . . even Intel stock didn't have a too bad year

. . . even Intel stock didn't have a too bad year

How do these attacks work, in general?

. . . Side-channel attacks

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

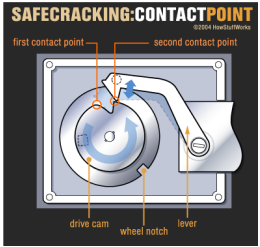
Attacker

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

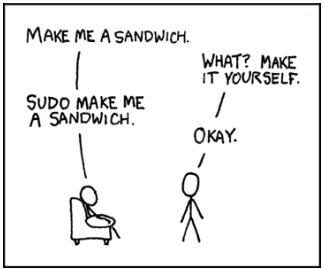


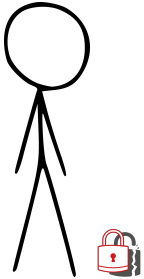
Figure: source: `xkcd.com/149/`

Luckily, these attacks can “only” read privileged memory

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

Attacker

Victim



action: none →

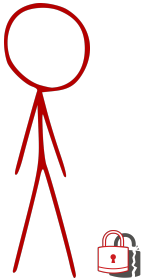
← carrier: cache changes

Foreshadow-OS

Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

Attacker



Foreshadow-OS

action: none →

← carrier: cache changes

Victim

Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

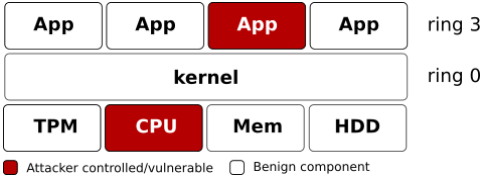
Setting: Attacker-controlled process

Attack model:

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

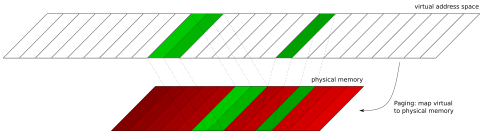
Attack objective:

- Read data outside the process' address space

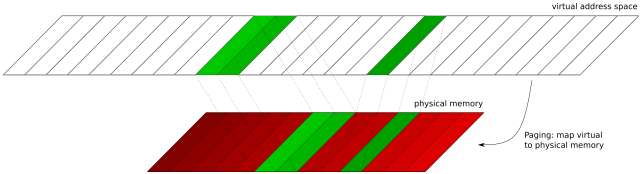


Background: How does process isolation work. . .

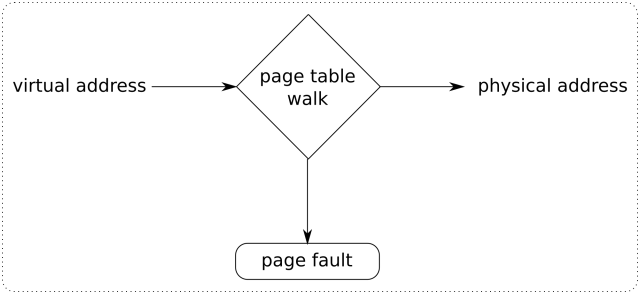
- MMU: map virtual address space to physical memory
- Protect physical memory by:
  - Not providing a mapping
  - Restricting access (e.g., U/S-bit)



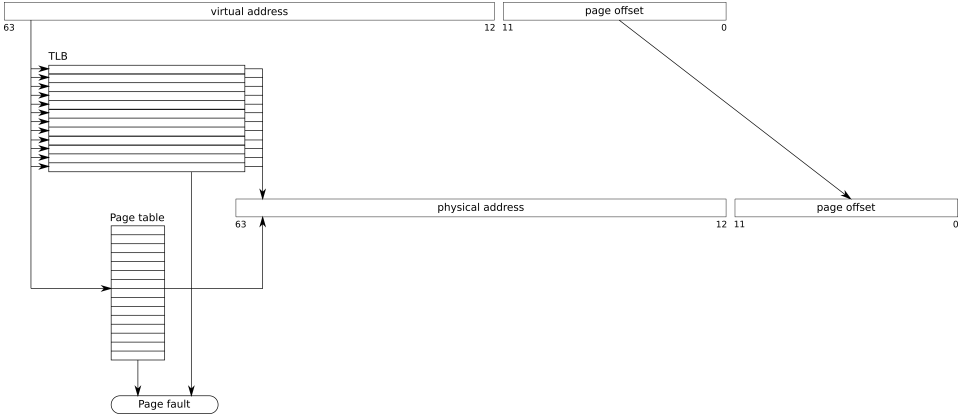
Background: How does process isolation work. . .



Background: How does process isolation work. . .



Background: How does process isolation work. . .



Background: How does process isolation work. . .

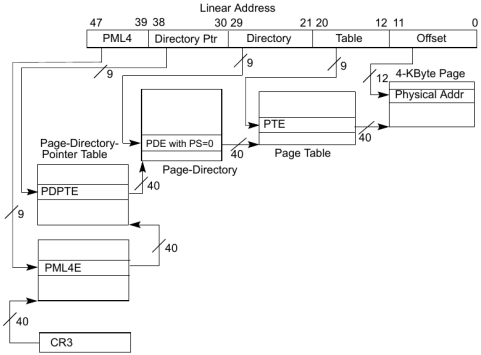


Figure: source: Intel 64 and IA-32 architectures software developer's manual

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. . .

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

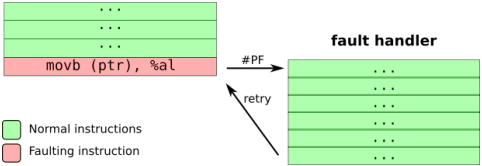
Background: Swapping memory pages to disk. . .

- Memory requirements may be larger than available physical memory
- Pages may be swapped-out to disk
- Swapping pages out:
  1. Mark virtual page as not-present
  2. Write physical page to swap space
  3. Mark physical page frame as free memory

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)

Background: Swapping memory pages to disk. . .

- Swapping pages in:
  1. Process accesses not-present page → #PF
  2. Read physical page from swap space
  3. Mark page as present
  4. Resume process



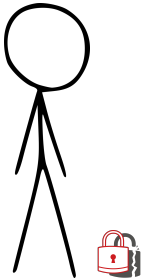
Background: Swapping memory pages to disk. . .

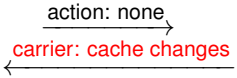
- Swapping pages in:
  1. Process accesses not-present page → #PF
  2. Read physical page from swap space
  3. Mark page as present
  4. Resume process

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

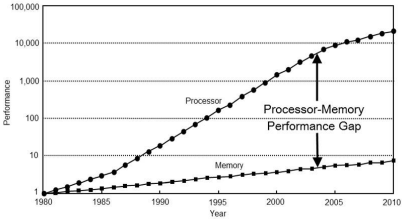
Other process' memory

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

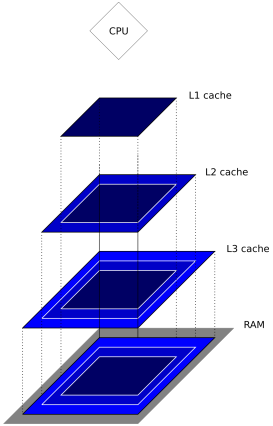


The message carrier: how does the cache work?

- Skylake: caches are inclusive
- Fetches from closest cache

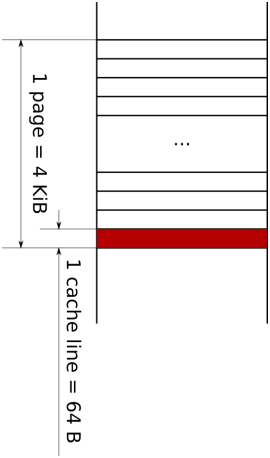
Cache	Latency	Capacity
L1D	4 cycles	32 KiB
L2	12 cycles	256 KiB
L3	44 cycles	2 MiB
RAM	~190 cycles	~8 GiB

Table: source: Intel 64 and IA-32 Architectures Optimization Reference Manual



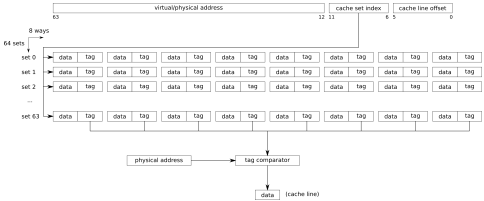
The message carrier: how does the cache work?

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

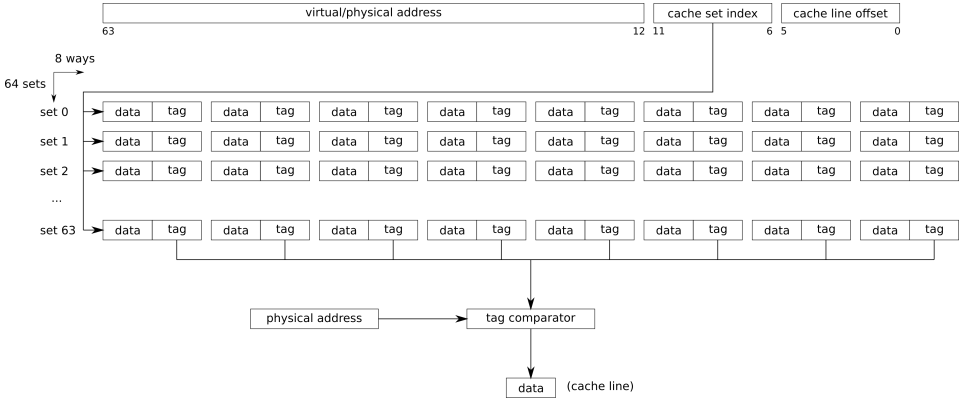


The message carrier: how does the cache work?

- 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

```
void time_cache_hit( uint64_t *timings, int num_runs ) {
  uint64_t *memory;
  volatile uint64_t tmp;
  uint64_t start, stop;

  // allocate memory

  for ( int run = 0; run < num_runs; ++run ) {
    // place memory in L1 cache
    tmp = *memory;

    // time accesses to RAM
    start = timestamp();
    tmp = *memory;
    end = timestamp();

    timings[run] = end - start;
  }
}
```

The message carrier: How does the cache work?

Manipulating the cache:

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

```
void time_cache_miss( uint64_t *timings, int num_runs ) {
  uint64_t *memory;
  volatile uint64_t tmp;
  uint64_t start, stop;

  // allocate memory

  for ( int run = 0; run < num_runs; ++run ) {
    // evict memory from all cache levels
    clflush( memory );

    // time accesses to RAM
    start = timestamp();
    tmp = *memory;
    end = timestamp();

    timings[run] = end - start;
  }
}
```

The message carrier: How does the cache work?

Manipulating the cache:

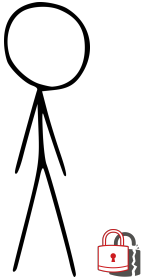
- Data accesses: load in L1-L3 cache
- `clflush`: Flush data from caches

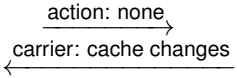
memory	timing (in cycles)	std. dev.
L1	46	1.25
L2	53	1.14
RAM	246	6.22

→ Any timing results <146 cycles clearly hits the cache

Attacker

Victim





Foreshadow-OS

Other process' memory

Security flaw: OoO execution leaves traces of transient instructions

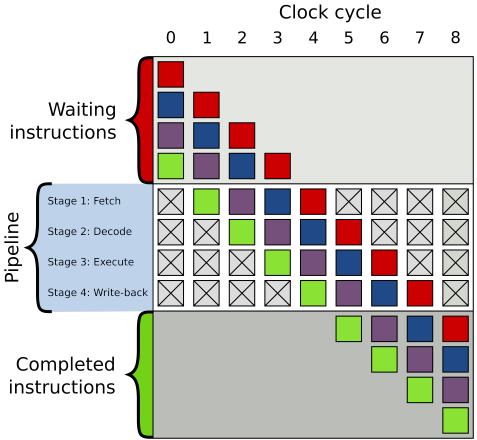
Background: Pipelining & Out of Order Execution

- **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: Pipelining & Out of Order Execution

Instruction pipelining:

- Split instructions in Fetch-Decode-Execute-Write Back phases
- 4 instructions are being processed at a time
- Upon fault: flush the pipeline



Background: Pipelining & 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

Knock, knock,

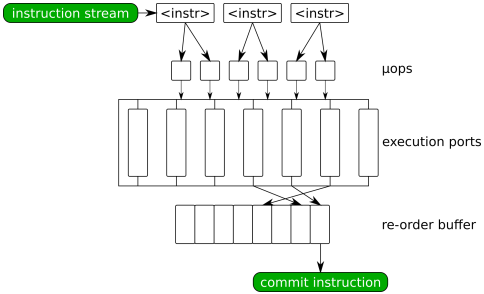
transient execution

who's there?

Background: Pipelining & Out of Order Execution

Out-of-order execution

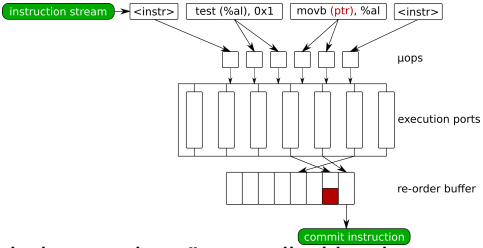
- Split instruction in μops
- Use multiple execution ports
- Execute μop as soon as possible
- Reorder ensures results/exceptions are visible in-order of instruction stream



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

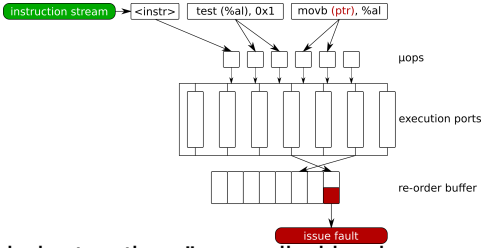


Key issue: Not all side-effects of “unreachable instructions” are rolled back correctly! (e.g., cache changes)

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

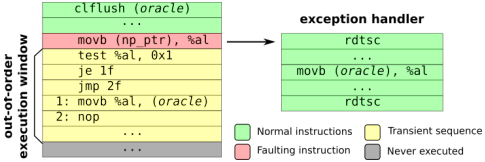


Key issue: Not all side-effects of “unreachable instructions” are rolled back correctly! (e.g., cache changes)

The Security Flaw: Transient Execution

Transient execution:

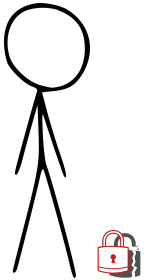
- Faults are detected at last moment
- Instruction that should *never* be executed, may already have started
- Processor rolls back architectural changes



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

Attacker

Victim



action: none →

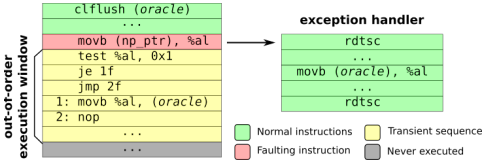
← carrier: cache changes

Foreshadow-OS

Other process' memory

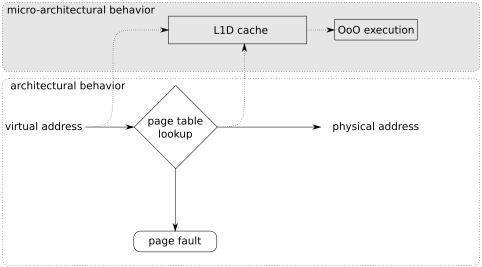
Security flaw: OoO execution leaves traces of transient instructions

Putting it all together



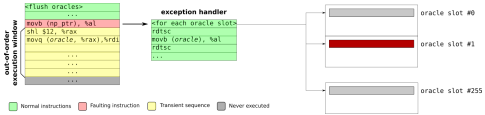

```
int8_t *oracle = ...;
int8_t *np_ptr = ...;

// Step 1: Remove variable oracle from cache
clflush( oracle );

// Step 2: Trick system in sensitive data in L1 but PTE present bit to 0

// Step 3: attempt to read not present memory
if ( *np_ptr == 1 )
  // place oracle variable in the cache iff *np_ptr == 1
  _tmp = *oracle;

// suppress fault

// Step 4: is oracle cached?
if ( time_access( oracle ) < 146 )
  print( "sensitive data == 1!" );
else
  print( "sensitive value != 1" );
```

Putting it all together



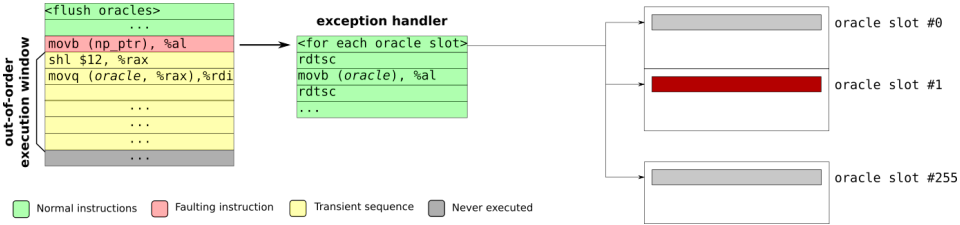

```
int8_t *oracle = ...;
int8_t *np_ptr = ...;

// Step 1: Remove variable oracle from cache
clflush( oracle );

// Step 2: Trick system in sensitive data in L1 but PTE present bit to 0

// Step 3: attempt to read not present memory
if ( *np_ptr == 1 )
  // place oracle variable in the cache iff *np_ptr == 1
  _tmp = *oracle;

// suppress fault

// Step 4: is oracle cached?
if ( time_access( oracle ) < 146 )
  print( "sensitive data == 1!" );
else
  print( "sensitive value != 1" );
```

Increasing the bandwidth of the attack




```
int8_t *oracles = ...;
int8_t *np_ptr = ...; // the secret
int8_t _tmp;

// Step 1: Remove oracle slots from cache
for ( int i = 0; i < 256; ++i )
  clflush( &oracles[4096 * i] );

// Step 2: Trick system in sensitive data in L1 but PTE present bit to 0

// Step 3: attempt to read not present memory
_tmp = oracle[4096 * (*np_ptr)];

// suppress fault

// Step 4: which oracle slot is cached?
for ( int i = 0; i < 256; ++i ) {
  if ( time_access( oracle[4096 * i] ) < 146 )
    print( "*np_ptr = %i\n", i );
}
```

Increasing the bandwidth of the attack



Who's Affected?

Vulnerable processors:

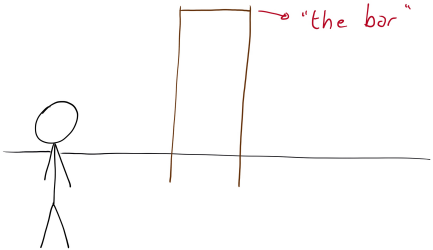
- Intel Core processors of the last 7 years
- Intel server processors
- NOT AMD, not ARM

Impact of this attack

Requirements:

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

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

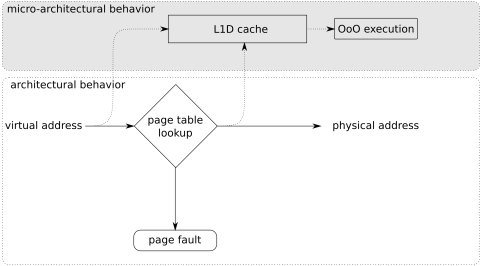


Mitigations

- 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

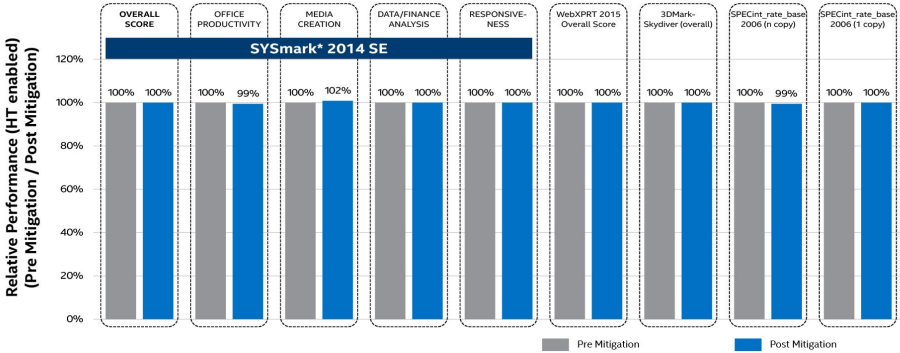
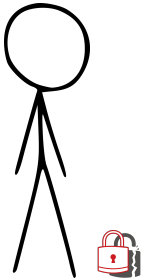


Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

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

Attacker

Victim



action: manipulate EPT →

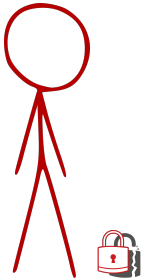
← carrier: cache changes

Foreshadow-VMM

Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Attacker



Victim

action: manipulate EPT →

← carrier: cache changes

Foreshadow-VMM

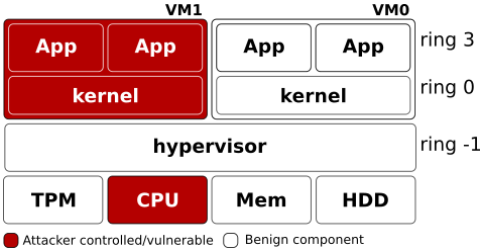
Other VM's memory

Security flaw: OoO execution leaves traces of transient instructions

Setting: Attacker-controlled VM

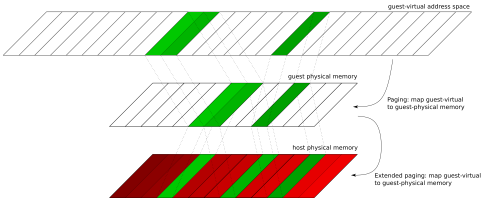
- Multiple VMs on one physical server
- Attacker-controlled VM
- Hypervisor ensures VM isolation

⇝ 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

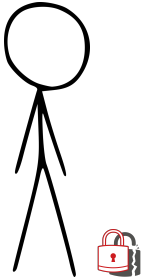


How do extended page tables work

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

Bit Position(s)	Contents
0	Read access; indicates whether reads are allowed from the 4-KByte page referenced by this entry
1	Write access; indicates whether writes are allowed from the 4-KByte page referenced by this entry
2	If the "mode-based execute control for EPT" VM-execution control is 0, execute access; indicates whether instruction fetches are allowed from the 4-KByte page controlled by this entry If that control is 1, execute access for supervisor-mode linear addresses; indicates whether instruction fetches are allowed from supervisor-mode linear addresses in the 4-KByte page controlled by this entry
5:3	EPT memory type for this 4-KByte page (see Section 28.2.6)
6	Ignore PAT memory type for this 4-KByte page (see Section 28.2.6)
7	Ignored
8	If bit 6 of EPTP is 1, accessed flag for EPT; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 28.2.4). Ignored if bit 6 of EPTP is 0
9	If bit 6 of EPTP is 1, dirty flag for EPT; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 28.2.4). Ignored if bit 6 of EPTP is 0
10	Execute access for user-mode linear addresses. If the "mode-based execute control for EPT" VM-execution control is 1, indicates whether instruction fetches are allowed from user-mode linear addresses in the 4-KByte page controlled by this entry. If that control is 0, this bit is ignored.
11	Ignored
(N−1):12	Physical address of the 4-KByte page referenced by this entry[1]
51:N	Reserved (must be 0)
62:52	Ignored
63	Suppress #VE. If the "EPT-violation #VE" VM-execution control is 1, EPT violations caused by accesses to this page are convertible to virtualization exceptions only if this bit is 0 (see Section 25.5.6.1). If "EPT-violation #VE" VM-execution control is 0, this bit is ignored.

Attacker



Victim

action: manipulate EPT →

← carrier: cache changes

Foreshadow-VMM

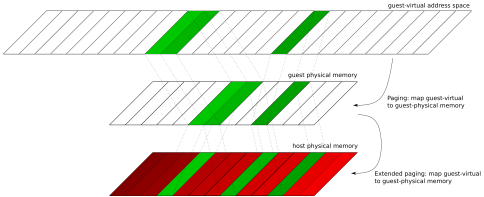
Other VM's memory

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!!

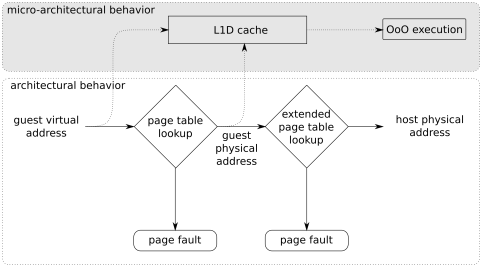
$\rightsquigarrow$ This breaks the VM's address space abstraction!



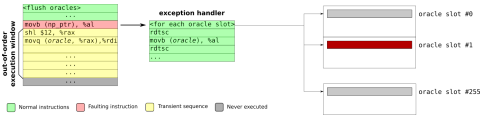
The Security Flaw: Interpreting guest-physical as host-physical addresses

- 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



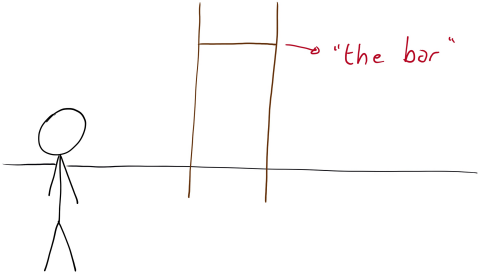

```
int8_t *oracles = ...;
int8_t *np_ptr = ...;
int8_t _tmp;

// Step 1: Setup PT to physical address of interest

// Step 2: Remove oracle slots from cache
for ( int i = 0; i < 256; ++i )
  clflush( &oracles[4096 * i] );

// Step 3: Wait for sensitive data in L1D

// Step 4: attempt to read not present memory
_tmp = oracle[4096 * (*np_ptr)];

// suppress fault

// Step 5: is oracle cached?
for ( int i = 0; i < 256; ++i ) {
  if ( time_access( oracle[4096 * i] ) < 146 )
    print( "*np_ptr = %i\n", i );
}
```

Impact of this attack

Requirements:

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

⇝ Modest impact!

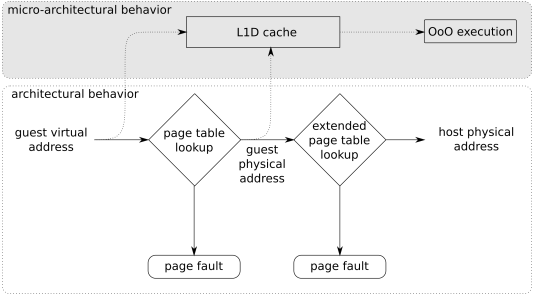


Impact of this attack

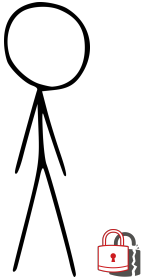
Requirements:

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

⇝ Modest impact!



Attacker



Victim

action: manipulate EPT →

← carrier: cache changes

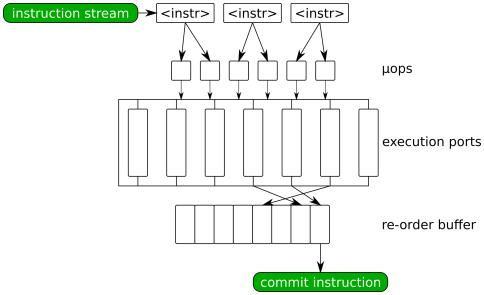
Foreshadow-VMM

Other VM's memory

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)



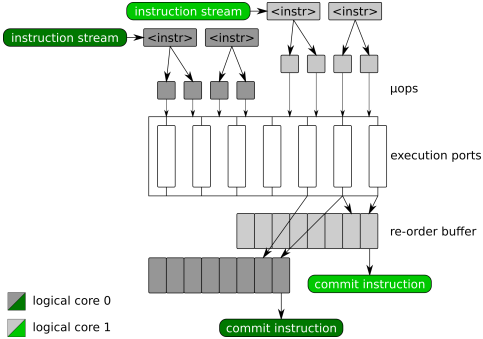
⇝ Performance increase of up to 30%[1]

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]



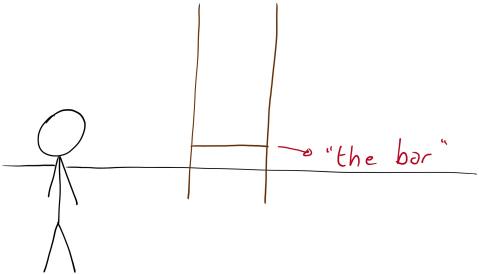
https://www.cs.sfu.ca/~fedorova/Teaching/CMPT886/Spring2007/papers/hyper-threading.pdf

Impact of this attack

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

Mitigations – Disabling HyperThreading

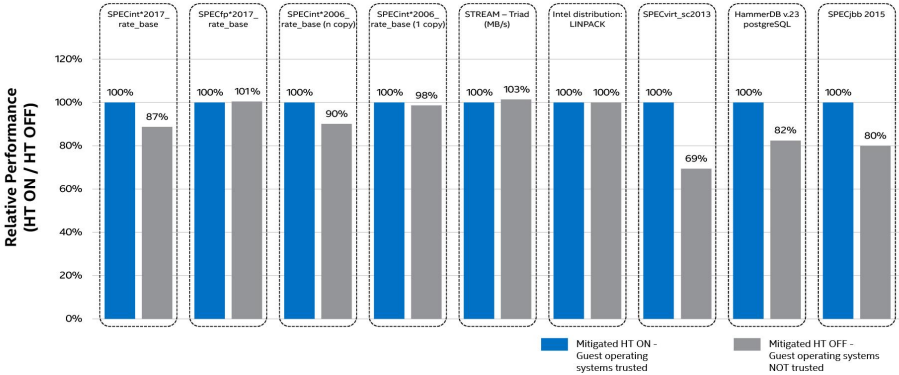


Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

Mitigations – Updating VM scheduler

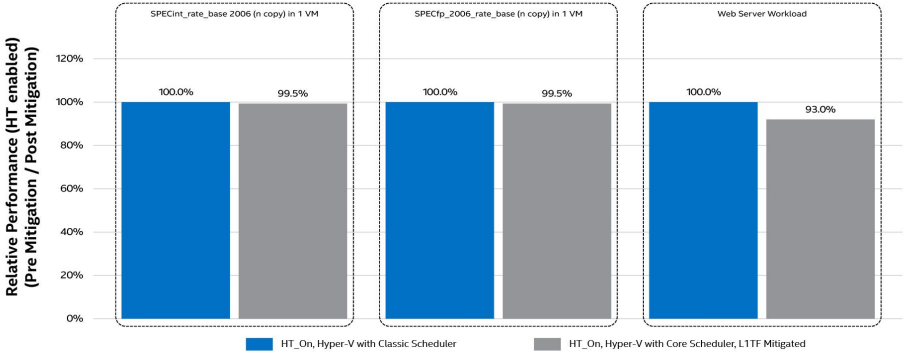
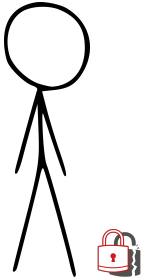


Figure: source:
`https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html`

Foreshadow-SGX: Dismantling Intel SGX's security objectives

Attacker



Victim

action: manipulate PT →

← carrier: cache changes

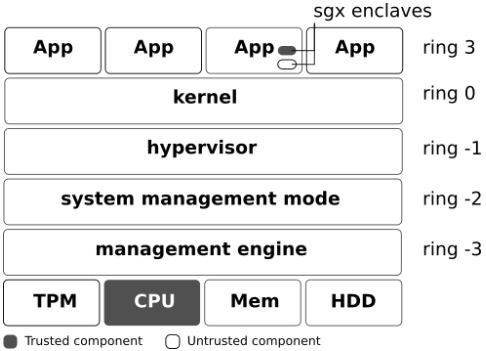
Foreshadow-SGX

SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

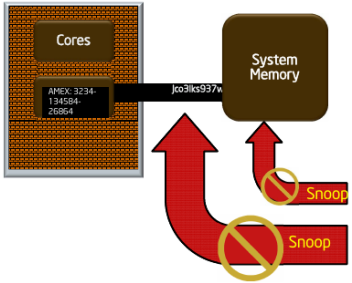
Background: Intel SGX

- **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

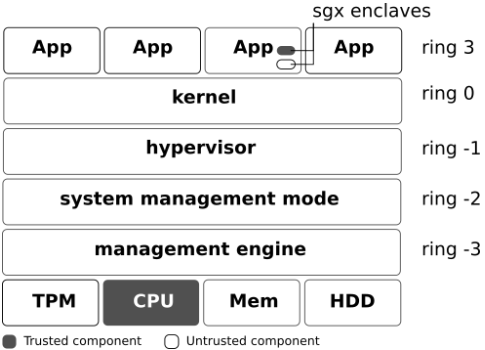
- **Problem**: Huge software TCB
- **Solution**: Protected-Module Architecture (e.g., Intel SGX)
- Only trust Intel hardware/enclaves
- Use cases:
  - protecting finger prints
  - DRM
  - Secure cloud-based processes
  - . . .



Background: Intel SGX

Key properties:

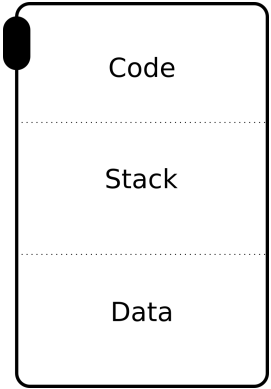
- Isolation
- Secure storage
- Attestation



Background: Intel SGX

Isolation:

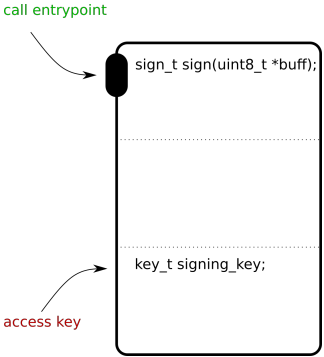
- Enclaves live in process' address space
- Only accessible through specific entry points
- Abort page semantics: Reading enclave memory outside the enclave results in `-1`.



Background: Intel SGX

Isolation:

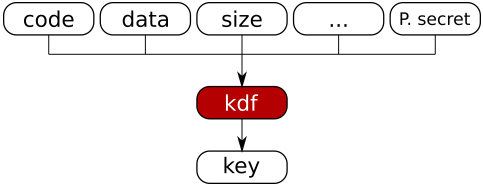
- Enclaves live in process' address space
- Only accessible through specific entry points
- Abort page semantics: Reading enclave memory outside the enclave results in `-1`.



Background: Intel SGX

Secure Storage:

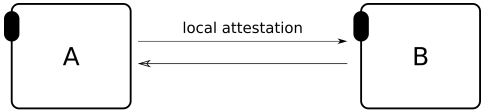
- Enclave die at loss of power
- Seal/Unseal confidential data
- Key derivation ensure unique key per enclave



Background: Intel SGX

Attestation:

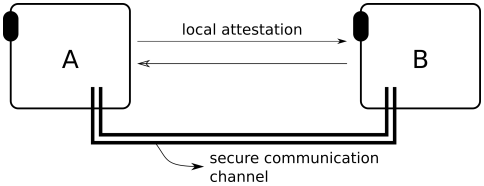
- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as building block for remote attestation
- EPID attestation protocol can ensure that attestation responses cannot be linked



Background: Intel SGX

Attestation:

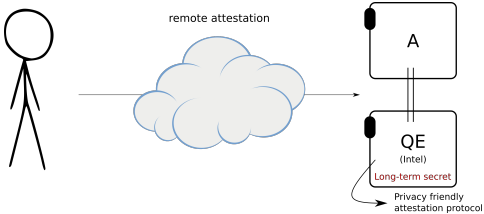
- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as building block for remote attestation
- EPID attestation protocol can ensure that attestation responses cannot be linked



Background: Intel SGX

Attestation:

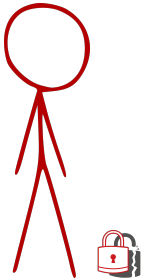
- Prove an enclave has been created correctly
- Both locally as remotely
- Local attestation as building block for remote attestation
- EPID attestation protocol can ensure that attestation responses cannot be linked



Background: Intel SGX

Figure: source: `xkcd.com/1957/`

Attacker



Foreshadow-SGX

action: manipulate PT →

← carrier: cache changes

Victim

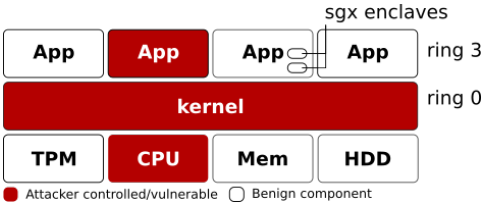
SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

Setting: Attacker-controlled machine

Attack model:

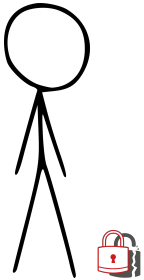
- Software-based attacker at any privilege level
- Hardware-based attacker outside CPU package

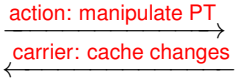


Note: The same attack also works in a VM

Attacker

Victim





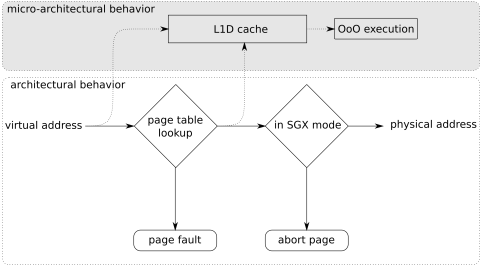
Foreshadow-SGX

SGX enclave memory

Security flaw: OoO execution leaves traces of transient instructions

The attack approach

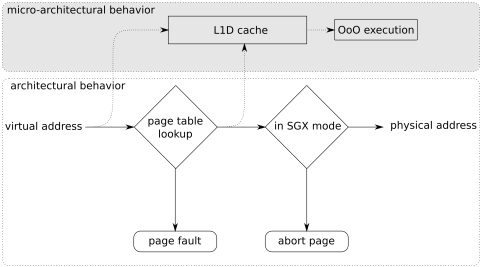
- Bypass abort page semantics
- Ensure data in L1D:
  - Zero-step through enclave
  - Some instructions load enclave data in L1D as a side effect (e.g., `eldu`)



Foreshadow-SGX: The Exploit


```
int8_t *oracles = ...;
int8_t *np_ptr = ...;
int8_t _tmp;

// Step 1: Set up enclave
...

// Step 2: Remove access rights to enclave
mprotect( np_ptr, 0x1000, PROT_NONE );

// Step 3: Setup cache
for ( int i = 0; i < 256; ++i )
  clflush( &oracle[4096 * i] );

// Step 4: Load enclave data in L1D
eldu( np_ptr );

// Step 5: attempt to read not present memory -- suppress fault
_tmp = oracle[4096 * (*np_ptr)];

// Step 6: is oracle cached?
for ( int i = 0; i < 256; ++i ) {
  if ( time_access( oracle[4096 * i] ) < 146 )
    print( "*np_ptr = %i\n", i );
}
```

Foreshadow-SGX: The Exploit



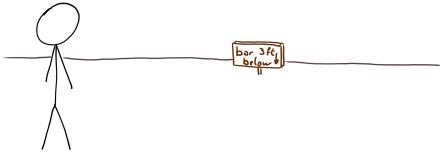

```
int8_t *oracles = ...;
int8_t *np_ptr = ...;
int8_t _tmp;

// Step 1: Set up enclave
...

// Step 2: Remove access rights to enclave
mprotect( np_ptr, 0x1000, PROT_NONE );

// Step 3: Setup cache
for ( int i = 0; i < 256; ++i )
  clflush( &oracle[4096 * i] );

// Step 4: Load enclave data in L1D
eldu( np_ptr );

// Step 5: attempt to read not present memory -- suppress fault
_tmp = oracle[4096 * (*np_ptr)];

// Step 6: is oracle cached?
for ( int i = 0; i < 256; ++i ) {
  if ( time_access( oracle[4096 * i] ) < 146 )
    print( "*np_ptr = %i\n", i );
}
```

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

Comparing Foreshadow/Meltdown/Spectre

	Title	CVE	SA	Severity	Disclosure Date
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`

Comparing Foreshadow/Meltdown

Meltdown: Similar to Foreshadow, but

- ...discovered independently
- ...also exploits transient instructions
- ...leaks data “passed” U/S bit
- ...does not require data to be cached
- ...“only” affects process/kernel isolation (not VM/VMMs)
- ...mitigated by not mapping full memory in process’ address space

Comparing Foreshadow/Spectre

Spectre:

- ...exploits mispredicted jumps
- ...leaks data from the process containing the jump
- ...much harder to execute
- ...much more difficult to mitigate

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

→ I’m more worried about the next big speculative execution attack (e.g., not based on cache usage)

Outlook: Long-term solutions

- Speculative execution cannot be removed completely without a significant performance hit
- Expose microarchitecture to compiler to reduce performance hit?
- Focus on parallel programming instead?

Figure: source: `xkcd.com/1312/`

Conclusion

- Foreshadow-VMM breaks the virtual memory abstraction
- Foreshadow-SGX completely dismantled SGX' security framework
- Modern x86 processors have become too complex to completely understand
- Need to completely rethink past design decisions

"...this paper serves as a helpful demonstration that SGX [...] rely not only on trusting Intel's beneficence but also their competence."

– Reviewer E, Usenix Security '18

Thank you!

Thank you! Questions?

raoul.strackx@cs.kuleuven.be
@raoul_strackx