Foreshadow
Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution

Ofir Weisse
Joint work with
Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, Raoul Strackx
'Foreshadow' attack affects Intel chips

Beyond Spectre: Foreshadow, a new Intel security problem

* Read about Ubuntu updates for L1 Terminal Fault Vulnerabilities (L1TF).
- https://ubuntu.com/security/l1tf
Foreshadow (SGX)

Confidentiality
Secure storage
Proof of integrity

Untrusted OS/VMM

Foreshadow-NG

VM1
VM2

Cloud Host
Evolution of Side Channel Attacks

Classic Cache Timing
(Algorithm specific, e.g., AES)

Spectre & Meltdown
(on own address space)

Foreshadow
(on others’ address space)
Roadmap

- Cache side channels
- Speculative execution
- Meltdown
- SGX
- Foreshadow-SGX
- Foreshadow-NG
Side Channel Attacks – Abusing Non-standard Output Channels
Cache Side Channels
Cache Hierarchy

- **L1 $**: ~4 cycles
- **L2 $**: ~12 cycles
- **L3 $**: ~60-80 cycles
- **Slow Memory, 128GB, 300-400 cycles to access**
Background: Cache Timing Side Channel

- Attacker infers victim’s data access pattern
- Attack is algorithm specific
Roadmap

• Cache side channels
• **Speculative execution**
• Meltdown
• SGX
• Foreshadow-SGX
• Foreshadow-NG
Speculative Execution

data = *user_input;
res = 42 / data;
b -= res;
b++;
c[0] *= 2;
d[1] += 42;

Speculating future tasks

Retired instruction
Speculative Execution

\[
data = \ast\text{user_input};
\]
\[
res = 42 / data;
\]
\[
b -= res;
\]
\[
b++;
\]
\[
c[0] *= 2;
\]
\[
d[1] += 42;
\]
Speculative Execution

```c
*user_input = 0;
data = *user_input;
res = 42 / data;
b -= res;
b++;
c[0] *=2;
d[1] += 42;
```

Squashed instructions may leave footprints in cache

Exception handler (division by 0)

Retired instruction

Pending instruction

Squashed instruction
Roadmap

- Cache side channels
- Speculative execution
- Meltdown
- SGX
- Foreshadow-SGX
- Foreshadow-NG
Background: Meltdown

Attacker’s user-space code

```c
char probe[256*STEP];
clflush(probe);
secret = *kernel_addr;
probe[secret*STEP]++;
```

Process Virtual Memory

User Virtual Address Space

Kernel Virtual Address Space

Cache hit!
Virtual Address Space

Kernel-Space

User-Space

Process virtual memory pages

Physical memory frames
The Page Table

Virtual address bits:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>47</td>
<td>12</td>
<td>11</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused</td>
<td>Virtual page number</td>
<td>Page offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTE (Page Table Entry):

<table>
<thead>
<tr>
<th>Physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
</table>

Page attribute bits:

```c
char probe[256*STEP];
clflush(probe);
secret = *kernel_addr;
probe[secret*STEP]++;
```
Meltdown Mitigation - KPTI

During user-code execution: kernel memory is unmapped
Meltdown Mitigation - KPTI

Virtual address bits:

<table>
<thead>
<tr>
<th>63</th>
<th>47</th>
<th>12</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>Virtual page number</td>
<td>Page offset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page Table

No translation
Roadmap

• Cache side channels
• Speculative execution
• Meltdown
  • SGX
• Foreshadow-SGX
• Foreshadow-NG
SGX (Software Guard eXtensions)
SGX in a nutshell

User Space  Enclave

OS Kernel

VMM

SMM

RAM  HW  CPU

Attestation

Remote Client
SGX – Memory Organization

- Physical Memory
- Enclave Page Cache (EPC)
- EPC Metadata

Encrypted by Memory Encryption Engine (MEE)
SGX Abort Page Semantics

Physical Memory

Enclave Page Cache (EPC)

EPC Metadata

0xFF
SGX Abort Page Semantics

- No exception raised
- Writes are ignored
- Reads return 0xFF

```c
char probe[256*STEP];
clflush(probe)
secret = *enclave_addr;
probe[secret*STEP]++
```
SGX Abort Page Semantics
Roadmap

• Cache side channels
• Speculative execution
• Meltdown
• SGX
  • Foreshadow-SGX
  • Foreshadow-NG
Foreshadow –
Causing a Translation Terminal Fault

- Variant 1: Invalid PTE (Page Table Entry)
- Variant 2: Enclave to Enclave (E2E) rogue mapping
Foreshadow –
Causing a Translation Terminal Fault

Virtual address bits:

<table>
<thead>
<tr>
<th>63</th>
<th>47</th>
<th>12</th>
<th>11</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>Virtual page number</td>
<td>Page offset</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PTE (Page Table Entry):

<table>
<thead>
<tr>
<th>Physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
</table>

Poison PTE: Clear present bit
What happens when the translation faults?

Faulty PTE (Page Table Entry):

<table>
<thead>
<tr>
<th>Physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L1 $</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>~4 cycles</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Following a terminal fault (from Intel’s report):

- SGX memory checks are skipped (no 0xFF)
- Boundaries between VM and host are ignored
- System Management Mode (SMM) checks are skipped

Foreshadow Attack

**Malicious OS attacker code**

```c
PoisonPTE(enclave_addr);
char probe[256*STEP];
clflush(probe);
secret = *enclave_addr;
probe[secret*STEP]++;
```

**Micro-Architectural Behavior**

1. walk page table-get PFN
2. Verify translation OK
3. Fetch data from L1 cache
   - Terminal fault-
   - skip further checks
   - no abort page (0xFF)

**Cache Lines**

- PTE: Physical Frame Number (PFN)
- Misc.
- User/OS
- R/W
- Present

- L1 $ ~4 cycles

- Terminal fault-
- skip further checks
- no abort page (0xFF)
Only Data in L1 Cache is Exposed

- Following a “terminal fault” only data in L1 cache may be fetched.

L1 $  
~4 cycles

Slow Memory, 128GB, 300-400 cycles to access

But what if the attacker can bring data into L1 cache?
Maliciously Fetching Into L1 Cache

Physical Memory

Enclave Page Cache (EPC)
Small: ~93 MB

Regular/unsecure memory
Large: e.g., 32 GB

- The OS can “securely” page-out and page-in SGX pages
- On page-in - the decrypted data passes through L1 cache

Victim doesn’t need to run!!
Foreshadow in Action

SGX enclave initialized!
SGX enclave: secret string received and stored safely in enclave memory!
SGX enclave: secret string at 0x7f19ee646000

Press enter to naively read enclave memory at address 0x7f19ee646000...
Segment 0: 0x7f19ee646000 - 0x7f19ee646317
Victim address = 0x7f19ee646316... 0xFF
Actual success rate = 0/791 = 0.00 %
Press enter to use Foreshadow to read enclave memory at address 0x7f19ee646000 ...
Implications on SGX Enclaves and Ecosystem

- **Confidentiality** is completely gone:
  Foreshadow can dump *entire enclaves*

- At any given time, without the enclave running

- **Secure storage** is not safe:
  Foreshadow can extract SGX sealing (secure storage) keys

- **Proof of integrity (attestation)** can be forged:
  Foreshadow can extract secrets from
  - Intel Launch Enclave
  - Intel Quote Enclave

**Ramification:** a collapse of the attestation ecosystem
Security Quiz

If a machine was hacked, no one knows, and there is **no data** on it...
Remote Attestation:
Establishing Trust with Remote Enclaves

1. I am software running inside an SGX enclave. Key share: $a \cdot G$

2. Verify Quote with Intel

3. I believe you. Key share: $b \cdot G$

4. Session key: $ab \cdot G$

Takeaway: trust is based on the EPID key
EPID - Enhanced Privacy ID

- EPID mega feature – awesome privacy
- Millions of signatures are unlinkable
- No one knows who signed what

EPID failure – abusing privacy

A single extracted EPID key can be used to sign millions of unlinkable signatures

@ForeshadowAaaS
Foreshadow-SGX Mitigations

• Flush L1 Cache after enclave exits and “page-in/out” operations
  • New L1 flush “instruction” added
• Disable HyperThreading
• Have two sets of Attestation/Sealing keys
  • For HyperThreading On/Off
Roadmap

- Cache side channels
- Speculative execution
- Meltdown
- SGX
- Foreshadow-SGX

**Foreshadow-NG**
- User-space to kernel
- Reading SMM memory
- VM-to-VM/M
Nested Virtual Address Space

- Kernel-Space
- User-Space

Process virtual memory pages

Guest
Physical memory

Host
Physical memory
The Extended Page Table & Foreshadow

Controlled by the Malicious VM

Virtual address bits:

<table>
<thead>
<tr>
<th>Unused</th>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
</table>

Guest PTE:

<table>
<thead>
<tr>
<th>Guest</th>
<th>physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
</table>

Extended Page Table (EPT)

Host PTE:

<table>
<thead>
<tr>
<th>Host</th>
<th>physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
</table>
The Extended Page Table & Foreshadow

Controlled by the Malicious VM

Virtual address bits:

<table>
<thead>
<tr>
<th>Unused</th>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
</table>

Guest PTE:

- **Guest** physical frame number
- Misc.
- User/OS
- R/W
- Present

Host PTE:

- **Host** physical frame number
- Misc.

Guest physical address is treated as host physical address.
The Extended Page Table & Foreshadow

Controlled by the Malicious VM

Virtual address bits:

<table>
<thead>
<tr>
<th>Unused</th>
<th>Virtual page number</th>
<th>Page offset</th>
</tr>
</thead>
</table>

Guest PTE:

<table>
<thead>
<tr>
<th>Guest physical frame number</th>
<th>Misc.</th>
<th>User/OS</th>
<th>R/W</th>
<th>Present</th>
</tr>
</thead>
</table>

Guest physical address is treated as host physical address
Implications

• VM boundary is broken
• A malicious VM can read data from a neighboring VM or the VMM
Attack Limitations

- Data needs to reside in L1 cache (unlike the SGX attack)
- Attacker needs to guess/know physical address
- No known attacks in the wild
Mitigating Foreshadow-NG

- Disabling HyperThreading is devastating for performance
  - So what can we do?
- Never run two VMs on the same physical core
  - May impact performance
- Flush L1 cache on VMENTER
- On VMEXIT to hypervisor – make sure other sibling core is trusted
Conclusions

• Foreshadow-SGX: a complete break of SGX, including
  • Confidentiality
  • Secure storage
  • Attestation

  Patch your machine!

• Privacy-preserving protocols can backfire (e.g., EPID)
• Foreshadow-NG: VM boundary is cracked
• Mitigations come at a performance cost

ForeshadowAttack.com