2018: Meltdown Attack?

```c
char secret = *((char *) 0xffffffff81a0123);
printf("%c\n", secret);
```
char secret = *(char *) 0xffffffff81a0123;
char secret = *(char *) 0xffffffff81a0123;
char secret = *(char *) 0xffffffff81a0123;
2018: Meltdown Attack? (Step 1)

```c
char secret = *(char *) 0xffffffff81a0123;
```

![Diagram showing virtual address space, oracle, transient domain, CPU registers, and 256 different CPU cache line.](image)
2018: Meltdown Attack? (Step 2)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 2)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
char secret = *(char *) 0xfffffffff81a0123;
char x = oracle[secret * 4096];
2018: Meltdown Attack? (Step 3)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 3)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
2018: Meltdown Attack? (Step 3)

```c
char secret = *(char *) 0xffffffff81a0123;
char x = oracle[secret * 4096];
```
Microarchitecture Data Sampling (MDS)

• Meltdown is fixed but you can still leak on the fix hardware.

```c
char secret = *(char *) 0xfffwhateve123;
```

• Which part of the CPU leak the data?!

• Why does it leak?
CPU Memory Subsystem - Leaky Buffers

- **MSBDS**
- **MLPDS**
- **MFBDS**
- **L1TF**
Virtual Address

```plaintext
| VFN | Offset |
```
Virtual Address

| VFN | Offset |

PTE

| P | RW | US | ... | A | ... | Physical Page Number | ... |
Memory Access → Canonical → TLB → Perm. → Present

Virtual Address

- VFN
- Offset

PTE

- P
- RW
- US
- ... A ...
- Physical Page Number
- ...
Memory Access → Canonical → TLB → Perm. → Present → Accessed → Set A Bit

Virtual Address

```
| VFN | Offset |
```

PTE

```
P  RW  US  ...  A  ...  Physical Page Number  ...  
```

Set A Bit

Aligned Vector → #GP
Challenges with MDS Testing?

- Reproducing attacks is not reliable. It may depend on:
  - massaging the pipeline with other instructions
  - CPU configuration (generation, frequency, microcode patch and etc)
Challenges with MDS Testing?

• Reproducing attacks is not reliable. It may depend on:
  • massaging the pipeline with other instructions
  • CPU configuration (generation, frequency, microcode patch and etc)

• No public tool to find new variants or to verify hardware patches:
  • Too many things to test (Addressing mode, cache state, assists, and faults)
  • Previous POCs may not work after MC update, but what does it mean?
Challenges with MDS Testing?

• Reproducing attacks is not reliable. It may depend on:
  • massaging the pipeline with other instructions
  • CPU configuration (generation, frequency, microcode patch and etc)

• No public tool to find new variants or to verify hardware patches:
  • Too many things to test (Addressing mode, cache state, assists, and faults)
  • Previous POCs may not work after MC update, but what does it mean?

• Impossible to quantify the impact of leakage:
  • We should care about leakage rate and what data is leaked.
  • My POC is faster than your POC!!
Transynther
Transynther (Fuzzing-based Random MDS Testing)

Step 1:

\[
\text{char secret} = *(\text{char} \ *) \ 0xfffffffff81a0123;
\]

Step 2:

\[
\text{char } x = \text{oracle[secret } \times \ 4096];
\]

Step 3:

\[ 'P' = 0x50 \]

256 different CPU Cache Line
Transynther (Fuzzing-based Random MDS Testing)

Step 1:

Step 2:

$\text{char } x = \text{oracle[secret } \times 4096\text{];}$

Step 3:

$'P' = 0x50$

256 different CPU Cache Line
Transynther (Fuzzing-based Random MDS Testing)

Step 0: Buffer Grooming

Step 1:

Step 2:

Step 3:

char x = oracle[secret * 4096];

‘P’ = 0x50

256 different CPU Cache Line
Transynther (Fuzzing-based Random MDS Testing)

**Step 0:** Buffer Grooming

**Step 1:**

**Step 2:**

**Step 3:**

```
char x = oracle[secret * 4096];
```

`'P' = 0x50`

256 different CPU Cache Line
Transynther (Fuzzing-based MDS Testing)
Transynther (Fuzzing-based MDS Testing)
Transynther (Fuzzing-based MDS Testing)
<table>
<thead>
<tr>
<th>File Path</th>
<th>Branches</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>i3-7100_9_0x84_notsx.log_1_1835.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i3-7100_9_0x84_notsx.log_21829_473.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_17_1764.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_1_247.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_21829_1025.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_21829_1240.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_21829_2188.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_21829_2547.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0x48.log_21829_653.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0xcxa.log_21829_1175.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0xcxa.log_21829_541.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>i7-7700_9_0xcxa.log_21829_561.asm</td>
<td>init</td>
<td>3 days ago</td>
</tr>
<tr>
<td>Case</td>
<td>Preparation</td>
<td>Store</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>-------</td>
</tr>
<tr>
<td>①</td>
<td>(access ☐, random instructions)</td>
<td>-</td>
</tr>
<tr>
<td>②</td>
<td>(access ☐, random instructions)</td>
<td>-</td>
</tr>
<tr>
<td>③</td>
<td>(access ☐, random instructions)</td>
<td>-</td>
</tr>
<tr>
<td>④</td>
<td>(access ☐, random instructions)</td>
<td>store (to load)</td>
</tr>
<tr>
<td>⑤</td>
<td>(rep mov + store, store + fence + load)</td>
<td>store (to load)</td>
</tr>
<tr>
<td>⑥</td>
<td>(rep mov + store, store + fence + load)</td>
<td>store (4K Aliasing)</td>
</tr>
<tr>
<td>⑦</td>
<td>(rep mov + store, store + fence + load)</td>
<td>store (4K Aliasing, to load)</td>
</tr>
<tr>
<td>⑧</td>
<td>(Sibling on/off)</td>
<td>store (random address)</td>
</tr>
<tr>
<td>⑨</td>
<td>(Sibling on/off + clflush (store address))</td>
<td>store (Cache Offset of Load)</td>
</tr>
<tr>
<td>⑩</td>
<td>(Sibling on/off + repmov (to Load))</td>
<td>store (to Load)</td>
</tr>
<tr>
<td>⑪</td>
<td>(random instructions)</td>
<td>Store (Unaligned to Load)</td>
</tr>
<tr>
<td>⑫</td>
<td>(random instructions)</td>
<td>AVX Store (to Load)</td>
</tr>
<tr>
<td>⑬</td>
<td>(random instructions)</td>
<td>random fill stores</td>
</tr>
</tbody>
</table>

< ☐ Non-canonical Address Fault  ☐ Non-present Page Fault  ☐ Supervisor Protection Fault  ☐ AVX Alignment Fault

Access-bit Assist  ☐ Split-Cache Access Assist  ☐ Access without fault or Assist
MDS Attacks - Insights

- Almost any exception/assist can leak from any buffer
- The CPU must flush the pipeline before executing an assist.
- Upon an Exception/Fault/Assist on a Load, Intel CPUs:
  - Execute the load until the last stage.
  - Flush the pipeline at the retirement stage (Cheap Recovery Logic).
  - Continue the load with some data to reach the retirement stage.
- Which data? (Fill buffer, Store Buffer, Load Buffer)
- Which one will be leaked first? (First come first serve)
Medusa Attack

• Medusa only leaks the Write Combining Data
• Implicit WC, i.e., ‘rep mov’, ‘rep sto’, can be leaked.
  • Memory Copy Routines
  • File IO
• Served by a Write Combining Buffer (or just the Fill Buffer).

• Advantages:
  • Prefiltered data
  • Less Noise
  • More targeted
An invalid (Non-canon) address:
0x555000000000000008-20
Medusa Attack - V1 Cache Indexing

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

An invalid (Non-canon) address:
0x5550000000000008-20

Faulty Load
### Medusa Attack - V1 Cache Indexing

**Cache Line Index**

<table>
<thead>
<tr>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
<th>8-byte</th>
</tr>
</thead>
</table>

An invalid (Non-canon) address:

0x55500000000000008-20
Medusa Attack - V1 Cache Indexing

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Common Data Bus?!
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Faulty Load
Faulty Load

Cache Line Index

8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte

Faulty Load

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTUVWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Store

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSTU VWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Store

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOPQRSTU VWX YZ...
Medusa Attack - V2 Unaligned S2L Forwarding

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Store

Cache Line Index

| 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte | 8-byte |

Faulty Load

YMMx

REPMOV on the Hyper thread:

ABCDEFGH IJKLMNOP QRSU VWXYZ...
Medusa Attack - V3 Shadow REP MOV

- A REP MOV that fault on the load leaks:
  - the data from the legitimate store address
  - but also the data from the REP MOV running on the hyper thread
Medusa Attack - V3 Shadow *REP MOV*

• A *REP MOV* that fault on the load leaks:
  • the data from the legitimate store address
  • but also the data from the *REP MOV* running on the hyper thread

```
HT 1: REP MOV
Valid Store, Faulty Load

AAAAA........AAAAA
AAAAA........AAAAA

MD Leak

AAAAA........AAAAA
```

```
HT 1: REP MOV
Valid Store, Faulty Load

ABCDEFGIJKLMNOP
AAAAA........AAAAA

AAAAA........AAAAA...
```
Table 1: The performance counters used in Transynther to identify the active microarchitectural elements.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM_LOAD_RETIRED.FB_HIT</td>
<td>Data loaded from a line-fill-buffer entry.</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L1_HIT</td>
<td>Data loaded from the L1 data cache.</td>
</tr>
<tr>
<td>MEM_LOAD_RETIRED.L2_HIT</td>
<td>Data loaded from the L2 data cache.</td>
</tr>
<tr>
<td>LD_PENDING_MISS.FB_FULL</td>
<td>Data is neither in L1 nor in full buffer.</td>
</tr>
<tr>
<td>LD_BLOCKS_STORE_FORWARD</td>
<td>Store buffer blocks load.</td>
</tr>
<tr>
<td>LD_BLOCKS_PARTIAL_ADDRESS_ALIAS</td>
<td>Load blocked by partial address match.</td>
</tr>
<tr>
<td>MEM_INST_RETIRED.SPLIT_LOADS</td>
<td>Data spans across two cache lines.</td>
</tr>
</tbody>
</table>
• OpenSSL Base64 Decoder uses inline `Memcpy(-oS)`
• Triggered during the RSA Key Decoding from the PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjjtGtnlqMwmmaLW+YjbYTsaNR8PGKXr78iYwrMV5Ye4VGy
BwS6qLD4s/EzCzGIDwkwCVx+gVHvh2wGW15Ddofo0gVAAtAmkR6gRABy4TkK+6YFSK
AyjmHvKCFHjvc9loeFgDyjmwFFKfdwzppXnH1Wwt0OlncU1GQ1w7AHuwIDAQAB
AoGBAMyDri7pQ29NBIftMGMqFtw8wJ0R3EamIdqby7qUgUEoe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzd7oZ4qaeelv0F0+ZSz6CK7YbuzG2IXUB8nHJ7Nuh3lacfiyD
V4Cfgo0yFhTK+MDG/xTVqywrCTsslTCYCX/ZOXU5xt532FZAkEA/nLWQhMC4YPM
0LqMtgKzfgQdJ7vbr43WVVNpC/dn/ibUASI/3YwY0uUtqSjillghlY7pRohrPJ6W
ntSjw0UAhQJBAo2b9cFIOTFyKxyU4j315Vku1FFtyL6GwX/7mvpcDCixDLNryk
uRigmdKjtIUrAX0pwjgXa6niqJ691jExez8CQQcccMZZAvTbZhHsn9LhLxqS0IHY
K+ZzX5ogirFDP55QzzyE7adSsntSioh6/LQK8x6BAR9FwtxBPACtwz5F9geZAkA8
a3z0SlvG04aC1cjkGUPsx6wxxbl79F2RhmSKRbhv7JiYk3RQ+L7yJgmWPGu5AcLM
oVPjmbbkKfJZNTyVOW/AkABepEi++ZQQW0FXJWZ3nM+2CNCXYCtTgi4bGkvNZPp
/1pAy9rjeVJYhb8acTRnt+uUu74CTtfuzUTZLOluVe
-----END RSA PRIVATE KEY-----
```
OpenSSL RSA Key Recovery

- OpenSSL Base64 Decoder uses inline Memcpy(-oS)
- Triggered during the RSA Key Decoding from the PEM format:

```
-----BEGIN RSA PRIVATE KEY-----
MIICXQIBAAKBgQDmTvQjjtGtnlqMwmmaLM+YjbYTsNR8PGKXR78iYwrMV5Ye4VGyBwS6qLD4s/EzCzGIDwkWCVx+gVHvh2wGW15Ddofo0gVAtAmKr6gRABy4TkK+6YFSK
AyjmHvKcFHvc9loeFbGyjmwwFFkfdwzppXnH1Wwt00lnyCU1GbQ1w7AHuwlDAQAB
AoGBAMyDri7pQ29NBIfMnGQuFtw8c0R3EamIdQbX7qUguFEOe2YHqjdrKho5oZj
nDu8o+Zzm5jzBSzf7oZ4qaeekv0fO+ZS6CKYLBuzG2IXUB8nHJ7NuH3lacfiyDV4Cfg0yFnTK+MDG/xTVqywrCTsllkTCyC/XZOXU5Xt5z32FZAKcEA/nLWQhMC4YPM
0LqMtgKzfgQdJ7vbr43W2VvNpC/dn/iBuASl/3Yw70uUtqSjllghlY7pRohrPJK6W
ntSjw0UAhQJBAOe2b9cfiOTFKXxyU4j315VkuLFyl6GwXi/7mvpcDCixDLNRyk
uRigmdKjtULruAX0pwjgXa6niquJ691jExez8CQQCcMZZAvTbZHSh9LwHxq50SIY1
K+ZxX5ogirFDPS5NQzyE7adSsntSioh6/LQKBX6BAR9FwtxBPACtwzw5F9geZAkA8
a3z0SliVG04aC1cjkgUPsx6wwxb79F2RhsmSKRvh7JYK3RQ+L7VjgwmWPGu5AcLM
oVPsjmmbbkFkJZNTvW0V/AmABepEi++3QQW0FXJWZ3nM+2CNCXYCtTgi4bGkvnZPp
/1pAy9rjeVJYhbb8acTRnt+dU+uZ74CTtfuzUTZLOluVe
-----END RSA PRIVATE KEY-----
```
OpenSSL RSA Key Recovery

• OpenSSL Base64 Decoder uses inline Memcpy(-oS)
• Triggered during the RSA Key Decoding from the PEM format:

\[
\begin{align*}
N & \text{ (Modulus)} \\
\text{d (Private Key)}
\end{align*}
\]

\[
\begin{align*}
P \\
Q \\
d \mod (p-1) \\
d \mod (q-1) \\
Q^{\cdot(-1)} \mod p
\end{align*}
\]
OpenSSL RSA Key Recovery - Coppersmith

- Knowledge of at least $\frac{1}{3}$ of $P+Q$
- Create a $n$ dimensional hidden number problem where $n$ is relative to the number of recovered chunks
- Feed it to the lattice-based algorithm to find the short vector

\[
\begin{array}{c|c|c}
\hline
P & & \\
\hline
Q & & \\
\hline
\end{array}
\]
• Knowledge of at least \( \frac{1}{3} \) of \( P+Q \).
• Creating a \( n \) dimensional hidden number problem where \( n \) is relative to the number of recovered chunks.
• Feeding it to the lattice-based algorithm to find the short vector.

\[
\begin{align*}
P &
\end{align*}
\]

\[
\begin{align*}
Q &
\end{align*}
\]
Responsible Disclosure

• Medusa
  • June 24, 2019: Reported initial findings to Intel
  • Intel confirmed that WC is part of the fill buffer, but embargoed due to TAA
  • Nov 12, 2019: $$$ Awarded
Conclusion

• Automated Testing for CPU Attacks
  • helps us to understand the root cause of these issues better.
  • can be used to verify hardware mitigations.
  • can help us to improve the leakage rate and understand the impact of attacks better.

• The impact of attacks depend also on the exploitation technique.
Conclusion

• **Automated Testing for CPU Attacks**
  • helps us to understand the root cause of these issues better.
  • **can be used to verify hardware mitigations.**
  • can help us to improve the leakage rate and understand the impact of attacks better.

• The impact of attacks depend also on the exploitation technique.
Conclusion

• Automated Testing for CPU Attacks
  • helps us to understand the root cause of these issues better.
  • can be used to verify hardware mitigations.
  • can help us to improve the leakage rate and understand the impact of attacks better.

• The impact of attacks depend also on the exploitation technique.
Responsible Disclosure (Ice Lake)

• MSBDS (Fallout) on Ice Lake
  • November 2019: Intel sent us an Ice Lake Machine (Hardware mitigations)
Responsible Disclosure (Ice Lake)

• MSBDS (Fallout) on Ice Lake
  • November 2019: Intel sent us an Ice Lake Machine
  • March 2019: Tested Transyther on the Ice Lake CPU
Responsible Disclosure (Ice Lake)

• MSBDS (Fallout) on Ice Lake
  • November 2019: Intel sent us an Ice Lake Machine
  • March 2019: Tested Transsyther on the Ice Lake CPU

```asm
/** asm.S **/
.global s_faulty_load
s_faulty_load:
  lea address_normal+0x4822, %r14  // Store address
  lea address_supervisor +0x822, %r15 // Load address (4K alias)
  // clflush (%r14) // Uncomment to modify cache state
  // lock; incl (%r14) // Uncomment to modify cache state
  movb $0x41, (%r14) // Store
  movb (%r15), %al // Faulty Load
  and $0xff, %rax // Encode
  shlq $12, %rax
  movb (%r13,%rax,1), %al
ret
```
Responsible Disclosure (Ice Lake)

- **MSBDS (Fallout) on Ice Lake**
  - November 2019: Intel sent us an Ice Lake Machine
  - March 2019: Tested Transyther on the Ice Lake CPU
  - Mar 27, 2020: Reported MSBDS Leakage on Ice Lake
  - May 5, 2020: Intel Completed triage
    - MDS mitigations are not deployed properly
      - Chicken bits were not enabled for all mitigations.
      - OEMs shipped with old/wrong microcode.
    - Embargoed till July
  - July 13, 2020: MDS advisory and list of affected CPUs were updated.
    - $$$ Awarded
<table>
<thead>
<tr>
<th>MC Version</th>
<th>MC Date</th>
<th>Vulnerable</th>
<th>漏洞性能(bytes/s)</th>
<th>Unmodified</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x32 (stock)</td>
<td>2019-07-05</td>
<td>✓</td>
<td>elflush 577.87</td>
<td>754.99</td>
</tr>
<tr>
<td>0x36</td>
<td>2019-07-18</td>
<td>✓</td>
<td>lock inc 148.24</td>
<td>529.84</td>
</tr>
<tr>
<td>0x46</td>
<td>2019-09-05</td>
<td>✓</td>
<td></td>
<td>130.15</td>
</tr>
<tr>
<td>0x48</td>
<td>2019-09-12</td>
<td>✓</td>
<td></td>
<td>271.69</td>
</tr>
<tr>
<td>0x50</td>
<td>2019-10-27</td>
<td>✓</td>
<td></td>
<td>96.54</td>
</tr>
<tr>
<td>0x56</td>
<td>2019-11-05</td>
<td>✓</td>
<td></td>
<td>145.46</td>
</tr>
<tr>
<td>0x5a</td>
<td>2019-11-19</td>
<td>✓</td>
<td></td>
<td>532.40</td>
</tr>
<tr>
<td>0x66</td>
<td>2020-01-09</td>
<td>✗</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0x70</td>
<td>2020-02-17</td>
<td>✗</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0x82</td>
<td>2020-04-22</td>
<td>✗</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0x86</td>
<td>2020-05-05</td>
<td>✗</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 6: List of MDS-affected processors by Family/Model

<table>
<thead>
<tr>
<th>Family_Model</th>
<th>Step</th>
<th>Processor Families / Processor Number Series</th>
<th>MFBDS</th>
<th>MSBDS</th>
<th>MLPDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>06_7EH</td>
<td>5</td>
<td>10th Generation Intel® Core™ Processor Family based on Ice Lake (U, Y) microarchitecture</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

---

**057 MDS_NO Bit in IA32_ARCH_CAPABILITIES MSR is Incorrectly Set**

**Problem**
MDS_NO bit (bit 5) in IA32_ARCH_CAPABILITIES MSR (10Ah) is set, incorrectly indicating full activation of all MDS (microarchitectural data sampling) mitigations.

**Implication**
Due to this erratum, the IA32_ARCH_CAPABILITIES MDS_NO bit incorrectly reports the activation of all MDS mitigations actions.

**Workaround**
It is possible for the BIOS to contain a workaround for this erratum.

**Status**
For the steppings affected, refer to the Summary Table of Changes.
Questions?!

Daniel Moghimi
@danielmgmi

Moritz Lipp
@mlqxyz

Berk Sunar
@berksunar

Michael Schwarz
@misc0110

https://github.com/VernamLab/Medusa

https://github.com/danielmgmi/IceBreak