Microarchitectural Side-Channel Attacks

Michael Schwarz
September 18, 2019

www.iaik.tugraz.at
- Safe software infrastructure does not mean safe execution
• Safe software infrastructure does not mean safe execution
• Information leaks because of the underlying hardware
• Safe software infrastructure does not mean safe execution
• Information leaks because of the underlying hardware
• Exploit unintentional information leakage by side-effects
Side-channel Attacks

- Safe software infrastructure does not mean safe execution
- Information leaks because of the underlying hardware
- Exploit unintentional information leakage by side-effects

- Power consumption
- Execution time
- CPU caches
• Side channels also exist in software
• Side channels also exist in software
• Can be used for attacks
Side channels also exist in software
Can be used for attacks
Usually timing differences
Trivial approach: Compare each digit until a difference

```c
int check_pin(char* input) {
    const char* correct = "1234";
    for(int i = 0; i < 4; i++) {
        if(correct[i] != input[i]) {
            // digit differs, abort
            return ERROR;
        }
    }
    // PIN is correct
    return OK;
}
```
### Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
</tr>
</tbody>
</table>

- If digit is correct, next digit is checked → longer execution time
- 10 tries (maximum) to get a digit
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
</tbody>
</table>
• Measuring the **execution times** for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>
- Measuring the *execution times* for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
</tbody>
</table>
- Measuring the **execution times** for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td><img src="image" alt="0000 Time" /></td>
</tr>
<tr>
<td>1000</td>
<td><img src="image" alt="1000 Time" /></td>
</tr>
<tr>
<td>2000</td>
<td><img src="image" alt="2000 Time" /></td>
</tr>
<tr>
<td>3000</td>
<td><img src="image" alt="3000 Time" /></td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the *execution times* for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td><img src="image" alt="Bar Graph" /></td>
</tr>
<tr>
<td>1000</td>
<td><img src="image" alt="Bar Graph" /></td>
</tr>
<tr>
<td>2000</td>
<td><img src="image" alt="Bar Graph" /></td>
</tr>
<tr>
<td>3000</td>
<td><img src="image" alt="Bar Graph" /></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- If digit is correct, next digit is checked → longer execution time
Example: PIN Comparison

- Measuring the **execution times** for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- If digit is **correct**, next digit is checked → **longer** execution time
- 10 tries (maximum) to get a digit
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
</table>

• Repeat for every digit
• Longest execution time reveals correct digit
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
</tbody>
</table>
• Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td><img src="image" alt="1000 Bar" /></td>
</tr>
<tr>
<td>1100</td>
<td><img src="image" alt="1100 Bar" /></td>
</tr>
<tr>
<td>1200</td>
<td><img src="image" alt="1200 Bar" /></td>
</tr>
<tr>
<td>1300</td>
<td><img src="image" alt="1300 Bar" /></td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Repeat for every digit
Example: PIN Comparison

- Measuring the execution times for different PINs

<table>
<thead>
<tr>
<th>PIN</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td></td>
</tr>
<tr>
<td>1300</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- Repeat for every digit
- **Longest** execution time reveals correct digit
• Maximum 10 measurements per digit
Example: PIN Comparison

- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries
Example: PIN Comparison

- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries
- Brute force: 10,000 tries
Example: PIN Comparison

- Maximum 10 measurements per digit
- 4-digit PIN: 40 tries
- Brute force: 10,000 tries
- Simple side channel reduces tries by \textit{factor 250}
• Many functions can be implemented with constant runtime
Many functions can be implemented with **constant runtime**

```c
int check_pin(char* input) {
    const char* correct = "1234";
    int same = 0;
    for(int i = 0; i < 4; i++) {
        same |= correct[i] - input[i];
    }
    return (same == 0);
}
```
Example: PIN Comparison

- Many functions can be implemented with constant runtime

```c
int check_pin(char* input) {
    const char* correct = "1234";
    int same = 0;
    for(int i = 0; i < 4; i++) {
        same |= correct[i] - input[i];
    }
    return (same == 0);
}
```

- Sometimes, there is still a side channel in the hardware
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, …)
• Serves as the interface between hardware and software
• Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...) 
• Serves as the interface between hardware and software 
• Microarchitecture is an actual implementation of the ISA
Instruction Set Architecture (ISA) is an abstract model of a computer (x86, ARMv8, SPARC, ...)
Serves as the interface between hardware and software
Microarchitecture is an actual implementation of the ISA
• Modern CPUs contain multiple microarchitectural elements
- Modern CPUs contain multiple microarchitectural elements

Caches and buffer

Predictor
Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements
  - Caches and buffer
  - Predictor
  - Transparent for the programmer
Microarchitectural Elements

- Modern CPUs contain multiple microarchitectural elements

  - Caches and buffer
  - Predictor

- Transparent for the programmer
- Optimize program execution
• Modern CPUs contain multiple **microarchitectural elements**

- **Caches and buffer**
- **Predictor**

• **Transparent** for the programmer
• **Optimize** program execution
• **Timing differences** → side-channel leakage
printf("%d", i);
printf("%d", i);
Cache miss

printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
printf("%d", i);
CPU Cache

```c
printf("%d", i);
printf("%d", i);
```

Cache miss

Cache hit
CPU Cache

```
printf("%d", i);
printf("%d", i);
```

Cache miss

Cache hit

Request

Response

No DRAM access, much faster
CPU Cache

DRAM access, slow

Cache miss

printf("%d", i);
printf("%d", i);

Cache hit

No DRAM access, much faster
Memory Access Latency

Number of accesses

Access time [CPU cycles]

Cache Hits
Cache Misses
- L1 and L2 are private
- Last-level cache is
  - divided into slices
  - shared across cores
  - inclusive
Set-associative Last-level Cache

- Location in cache depends on the physical address of data

Memory Address

<table>
<thead>
<tr>
<th></th>
<th>11 bits</th>
<th>6 bits</th>
</tr>
</thead>
</table>

Cache

2048 cache sets
Location in cache depends on the physical address of data

Bits 6 to 16 determine the cache set
Set-associative Last-level Cache

- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set
- A cache set has multiple ways to store the data
- Location in cache depends on the physical address of data
- Bits 6 to 16 determine the cache set
- A cache set has multiple ways to store the data
- A way inside a cache set is a cache line, determined by the cache replacement policy
Flush+Reload

ATTACKER

flush

access

Shared Memory

cached

VICTIM

access

Shared Memory
Flush+Reload

ATTACKER

flush

access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush
access

Shared Memory

VICTIM

access
Flush+Reload

ATTACKER

flush

access

Victim accessed (fast)

Victim did not access (slow)

Shared Memory

VICTIM

access
struct shared_data[256];

[...] return shared_data[84];
[...]
struct shared_data[256];

return shared_data[84];

- Flush+Reload over memory locations
struct shared_data[256];

...]
return shared_data[84];
[...]

- Flush+Reload over memory locations

- Accessed index results in faster access time
Key presses trigger code execution in shared library (e.g., libgdk)
Key presses trigger code execution in shared library (e.g., libgdk).

Flush+Reload does not reveal actual key, only time difference between keys.

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks.
Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, Stefan Mangard. NDSS’18
• Key presses trigger code execution in shared library (e.g., libgdk)

• Flush+Reload does not reveal actual key, only time difference between keys

• → Recover text with machine learning

KeyDrown: Eliminating Software-Based Keystroke Timing Side-Channel Attacks. Michael Schwarz, Moritz Lipp, Daniel Gruss, Samuel Weiser, Clémentine Maurice, Raphael Spreitzer, Stefan Mangard. NDSS’18
string
A Double Fetch

```c
string

/ path / file \0 payload \0

length

Thread 1
strcpy (string, "/path/file\0payload");
open (string, O_CREAT);

Thread 2
```
```
A Double Fetch

string

/ path / file \0 payload \0

length

Thread 1

strcpy(string, "/path/file\0payload");
open(string, O_CREAT);

// <switch to kernel>

Thread 2
```
A Double Fetch

string

```
/path/file\0payload\0
```

length

Thread 1

```c
strcpy(string, "/path/file\0payload");
open(string, O_CREAT);

// <switch to kernel>

int len = strlen(string);
char* local = malloc(len + 1);
```
A Double Fetch

Thread 1

string

`strcpy (string, "/path/file\0payload");`

`open(string, O_CREAT);`

// <switch to kernel>

`int len = strlen(string);`

`char* local = malloc(len + 1);`

Thread 2

`schedule`

`string[10] = 'X';`
A Double Fetch

string

```
/ path / file X payload \0
```

Thread 1

```c
strcpy (string, "/path/file\0payload");
open(string, O_CREAT);

// <switch to kernel>

int len = strlen(string);
char* local = malloc(len + 1);
strcpy(local, string);

// <memory corruption>
```

Thread 2

```
string[10] = 'X';
```

Michael Schwarz — www.iaik.tugraz.at
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.

Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Double-fetch Detection

Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Cache-based Trigger

Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémentine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS’18
Automated Detection, Exploitation, and Elimination of Double-Fetch Bugs using Modern CPU Features.
Michael Schwarz, Daniel Gruss, Moritz Lipp, Clémantine Maurice, Thomas Schuster, Anders Fogh, Stefan Mangard. AsiaCCS'18
Prime+Probe

ATTACKER
prime
access

VICTIM
access
Prime+Probe

ATTACKER
prime
access

VICTIM
access
Victim did not access (fast) vs Victim accessed (slow)
\[ M = C^d \mod n \]
M = C^d \mod n
\[ M = C^d \mod n \]

\[ \begin{array}{c}
 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ \ldots \\
\end{array} \]

\[
\text{Result} = \text{Result} \times \text{Result} \times C
\]

\[ \text{square} \]

\[ \text{multiply} \]
\[ M = C^d \mod n \]

\[
\begin{array}{cccccccc}
1 & 1 & 0 & 0 & 1 & 1 & 0 & \cdots \\
\end{array}
\]

\[
\text{Result} = \text{Result} \times \text{Result}
\]

\text{square}
\[ M = C^d \mod n \]

Result = Result \times Result

\[ \text{square} \]
\[ M = C^d \pmod{n} \]

\[
\begin{array}{cccccccc}
\text{Result} & \times & \text{Result} & \times & C\\
\end{array}
\]

- square
- multiply
\[ M = C^d \mod n \]

Result = Result × Result × C

\[ \text{square} \quad \text{multiply} \]
\[ M = C^d \mod n \]

Result \[= \text{Result} \times \text{Result} \]

\[\text{square}\]
Raw Prime+Probe trace...
...processed with a simple moving average...

Malware Guard Extension: Using SGX to Conceal Cache Attacks.
Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, Stefan Mangard. DIMVA’17
...allows to clearly see the bits of the exponent
What is a *covert channel*?
- Two programs would like to communicate
What is a **covert channel**?

- Two programs would like to communicate but are **not allowed** to do so
What is a covert channel?

- Two programs would like to communicate but are **not allowed** to do so
  - either because there is no communication channel...
What is a **covert channel**?

- Two programs would like to communicate but are **not allowed** to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
What is a **covert channel**?

- Two programs would like to communicate but are **not allowed** to do so
  - either because there is no communication channel...
  - ...or the channels are monitored and programs are stopped on communication attempts
- Use **side channels** and stay stealthy
Covert channel
Covert channel
Sending Data

Last-level cache

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8
Sending Data

Sender

Last-level cache

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8

Receiver

evict

evict

evict

evict

evict

evict

evict

evict

Michael Schwarz — www.iaik.tugraz.at
Sending Data
Sending Data

Sender

Receiver

Last-level cache

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8

0
1
0
0
0
1
0
0

evict

evict
Sending Data

Last-level cache

Sender

0
1
0
0
1
0
0
0
0
0

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8

Receiver

measure → 0
measure → 1
measure → 0
measure → 0
measure → 1
measure → 0
measure → 0
measure → 0

Michael Schwarz — www.iaik.tugraz.at
Sending Data

Sender

Last-level cache

Receiver

0
0
1
0
1
0
0
0
1

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8
Sending Data

Sender

Last-level cache

Receiver

0
0
0
1
0
0
0
1
0
0
0
1
1
0
0
0
1
Evict

Cache Set #1
Cache Set #2
Cache Set #3
Cache Set #4
Cache Set #5
Cache Set #6
Cache Set #7
Cache Set #8

Evict

Evict

Evict

Evict
Sending Data

Last-level cache

- Cache Set #1
- Cache Set #2
- Cache Set #3
- Cache Set #4
- Cache Set #5
- Cache Set #6
- Cache Set #7
- Cache Set #8

Sender

- 0
- 0
- 1
- 0
- 1
- 0
- 0
- 1

Receiver

- measure → 0
- measure → 0
- measure → 1
- measure → 0
- measure → 1
- measure → 0
- measure → 0
- measure → 1
HELLO FROM THE OTHER SIDE (DEMO):
VIDEO STREAMING OVER CACHE COVERT CHANNEL
• Multiple other elements with timing differences
Other Microarchitectural Elements

- Multiple other elements with timing differences
  - TLB
• Multiple other elements with timing differences
  • TLB
  • DRAM
Other Microarchitectural Elements

- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
• Multiple other elements with timing differences
  • TLB
  • DRAM
  • Memory Bus
  • Execution Units
Other Microarchitectural Elements

- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units
  - ...

www.tugraz.at
Other Microarchitectural Elements

- Multiple other elements with timing differences
  - TLB
  - DRAM
  - Memory Bus
  - Execution Units
  - ...

- Many side-channel attacks exploiting them
• So far, only memory accesses
• So far, only memory accesses
• Meta data, no actual data
• So far, only memory accesses
• Meta data, no actual data
• Sufficient to deduce data...
- So far, only memory accesses
- Meta data, no actual data
- Sufficient to deduce data...
- ...if memory accesses are secret dependent
Side channels can be part of an attack
• Side channels can be part of an attack
• Also for *conventional* memory corruption attacks
• Side channels can be part of an attack
• Also for *conventional* memory corruption attacks
• Side channels as *building blocks*
• Side channels can be part of an attack
• Also for conventional memory corruption attacks
• Side channels as building blocks
  • Required information (e.g., break ASLR)
Side channels can be part of an attack

Also for conventional memory corruption attacks

Side channels as building blocks
  - Required information (e.g., break ASLR)
  - Additional information (e.g., length of password)
• Side channels can be part of an attack
• Also for conventional memory corruption attacks
• Side channels as building blocks
  • Required information (e.g., break ASLR)
  • Additional information (e.g., length of password)
  • Covertly transmit information
• Side channels can be part of an attack
• Also for conventional memory corruption attacks
• Side channels as building blocks
  • Required information (e.g., break ASLR)
  • Additional information (e.g., length of password)
  • Covertly transmit information
  • Transient-execution attacks
• Meltdown is a CPU vulnerabilities

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18
• Meltdown is a CPU vulnerabilities
• Discovered in 2017 by multiple independent teams

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security’18
Meltdown

- Meltdown is a CPU vulnerability
- Discovered in 2017 by multiple independent teams
- Allows breaking the process isolation

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18
Meltdown is a CPU vulnerabilities

Discovered in 2017 by multiple independent teams

Allows breaking the process isolation

Side-channel attack is a core building block

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18
• Kernel is isolated from user space
Hardware Isolation

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
Hardware Isolation

- Kernel is isolated from user space
- This isolation is a combination of hardware and software
- User applications cannot access anything from the kernel
- Kernel is isolated from user space
- This *isolation* is a combination of hardware and software
- User applications cannot access anything from the kernel
- There is only a well-defined interface → `syscalls`
In-Order Execution

- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
• Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
In-Order Execution

- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
  - executed (EX) by execution units
- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
  - executed (EX) by execution units
- Memory access is performed (MEM)
In-Order Execution

- Instructions are...
  - fetched (IF) from the L1 Instruction Cache
  - decoded (ID)
  - executed (EX) by execution units
- Memory access is performed (MEM)
- Architectural register file is updated (WB)
• Instructions are executed in-order
- Instructions are executed in-order
- Pipeline stalls when stages are not ready
In-Order Execution

- Instructions are executed in-order
- Pipeline stalls when stages are not ready
- If data is not cached, we need to wait
int width = 10, height = 5;

float diagonal = sqrt(width * width
+ height * height);

int area = width * height;

printf("Area \%d x \%d = \%d\n", width, height, area);
int width = 10, height = 5;

float diagonal = sqrt(width * width + height * height);
int area = width * height;

printf("Area %d x %d = %d\n", width, height, area);
Instructions are

- fetched and decoded in the front-end
Instructions are

- fetched and decoded in the **front-end**
- dispatched to the **backend**
Instructions are

- fetched and decoded in the front-end
- dispatched to the backend
- processed by individual execution units
Out-of-Order Execution

Instructions
- are executed out-of-order
Out-of-Order Execution

- Instructions are executed out-of-order.
- Instructions wait until their dependencies are ready.

Later instructions might execute prior to earlier instructions and retire in-order, with state becoming architecturally visible. Exceptions are checked during retirement. The pipeline can be flushed and state recovered if necessary.
Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
Out-of-Order Execution

Instructions

- are executed out-of-order
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire in-order
Out-of-Order Execution

Instructions

- are executed **out-of-order**
- wait until their dependencies are ready
  - Later instructions might execute prior earlier instructions
- retire **in-order**
  - State becomes architecturally visible
Out-of-Order Execution

Instructions

- are executed **out-of-order**
- wait until their **dependencies are ready**
  - Later instructions might execute prior earlier instructions
- retire **in-order**
  - State becomes architecturally visible
- **Exceptions** are checked during retirement
Out-of-Order Execution

Instructions

- are executed **out-of-order**
- wait until their **dependencies** are ready
  - Later instructions might execute prior earlier instructions
- retire **in-order**
  - State becomes architecturally visible
- Exceptions are checked during retirement
  - Flush pipeline and recover state
The state does not become architecturally visible but . . .
The state does not become **architecturally visible** but . . .
• New code

\[
\texttt{\ast (volatile char \ast) 0;}
\]

\[
\texttt{array[84 \ast 4096] = 0;}
\]
• New code

\[(volatile \ char\*) \ 0;\]
\[\text{array}[84 \ \times \ 4096] = 0;\]

• volatile because compiler was not happy

\text{warning: statement with no effect [−Wunused−value]}
\[(\text{char}*)0;\]
• New code

\[ *(\text{volatile char}*) \ 0; \]
\[ \text{array}[84 \times 4096] = 0; \]

• volatile because compiler was not happy

\text{warning: statement with no effect [–Wunused-value]}
\[ *(\text{char}*)0; \]

• Static code analyzer is still not happy

\text{warning: Dereference of null pointer}
\[ *(\text{volatile char}*)0; \]
• Flush+Reload over all pages of the array
• Flush+Reload over all pages of the array

• “Unreachable” code line was actually executed
Building the Code

- Flush+Reload over all pages of the array

- “Unreachable” code line was **actually executed**
- Exception was only thrown **afterwards**
• Out-of-order instructions leave microarchitectural traces
Out-of-order instructions leave microarchitectural traces

- We can see them for example in the cache
• Out-of-order instructions leave microarchitectural traces
  • We can see them for example in the cache
• Give such instructions a name: transient instructions
Out-of-order instructions leave microarchitectural traces
  - We can see them for example in the cache
Give such instructions a name: transient instructions
We can indirectly observe the execution of transient instructions
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
Loading an address
• Add another layer of indirection to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```
• Add another layer of indirection to test

```c
char data = *(char*) 0xffffffff81a000e0;
array[data * 4096] = 0;
```

• Then check whether any part of array is cached
- Flush+Reload over all pages of the array

- Index of cache hit reveals data
- Flush+Reload over all pages of the array

- Index of cache hit reveals data

- Permission check is in some cases not fast enough
Virtual address space

Kernel Direct-Physical Map

Physical memory

max. phys.

direct map

Kernel

User

$2^{47}$

$-2^{47}$
Using out-of-order execution, we can read data at any address.
• Using out-of-order execution, we can read data at any address
• Index of cache hit reveals data
• Using out-of-order execution, we can read data at any address
• Index of cache hit reveals data
• Permission check is in some cases not fast enough

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18
• Using out-of-order execution, we can read data at any address
• Index of cache hit reveals data
• Permission check is in some cases not fast enough
• Entire physical memory is typically accessible through kernel space

Meltdown: Reading Kernel Memory from User Space.
Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, Mike Hamburg. USENIX Security'18
I SHIT YOU NOT

THERE WAS KERNEL MEMORY ALL OVER THE TERMINAL
There are no bugs, just happy little accidents
• Meltdown is a whole category of vulnerabilities
• Meltdown is a whole category of vulnerabilities
• Not only the user-accessible check
Meltdown is a whole category of vulnerabilities.
Not only the user-accessible check.
Looking closer at the check...
• CPU uses **virtual address spaces** to isolate processes
- CPU uses **virtual address spaces** to isolate processes
- Physical memory is organized in **page frames**
• CPU uses virtual address spaces to isolate processes
• Physical memory is organized in page frames
• Virtual memory pages are mapped to page frames using page tables
Address Translation on x86-64

CR3

PML4
- PML4E 0
- PML4E 1
- PML4E 511

PDPT
- PDPTI 0
- PDPTI 1
- PDPTI 511

Page Directory
- PDE 0
- PDE 1
- PDE 511

Page Table
- PTE 0
- PTE 1
- PTE 511

4 KiB Page
- Byte 0
- Byte 1
- Offset
- Byte 4095

48-bit virtual address

Offset (12 b)

PML4I (9 b)  PDPTI (9 b)  PDI (9 b)  PTI (9 b)
- User/Supervisor bit defines in which **privilege level** the page can be accessed
<table>
<thead>
<tr>
<th>P</th>
<th>RW</th>
<th>US</th>
<th>WT</th>
<th>UC</th>
<th>R</th>
<th>D</th>
<th>S</th>
<th>G</th>
<th>Ignored</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

<table>
<thead>
<tr>
<th>Ignored</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>RW</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Physical Page Number**

Ignored

Ignored

X

- **Present** bit is the next obvious bit
Foreshadow-NG

- An even worse bug → Foreshadow-NG/L1TF

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security’18
• An even worse bug → Foreshadow-NG/L1TF
• Exploitable from VMs
An even worse bug → Foreshadow-NG/L1TF

- Exploitable from VMs
- Allows leaking data from the L1 cache

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security’18
Foreshadow-NG

- An even **worse** bug → Foreshadow-NG/L1TF
- Exploitable from **VMs**
- Allows **leaking** data from the **L1** cache
- Same mechanism as Meltdown

Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. USENIX Security’18
• An even worse bug → Foreshadow-NG/L1TF
• Exploitable from VMs
• Allows leaking data from the L1 cache
• Same mechanism as Meltdown
• Just a different bit in the PTE

### Page Table

<table>
<thead>
<tr>
<th>PTE 0</th>
<th>PTE 1</th>
<th>...</th>
<th>PTE #PTI</th>
<th>...</th>
<th>PTE 511</th>
</tr>
</thead>
</table>

![L1 Cache](image-url)
Page Table

| PTE 0 | PTE 1 | ... | PTE #PTI | ... | PTE 511 |

present

L1 Cache
Foreshadow-NG

Page Table

| PTE 0 |
| PTE 1 |
| ... |
| PTE #PTI |
| ... |
| PTE 511 |

present

Guest Physical to Host Physical

L1 Cache
Page Table

- PTE 0
- PTE 1
- ...
- PTE \#PTI
- ...
- PTE 511

Guest Physical to Host Physical

Physical Page

L1 lookup with physical address

L1 Cache
### Page Table

<table>
<thead>
<tr>
<th>PTE 0</th>
<th>PTE 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

PTE \#PTI

<table>
<thead>
<tr>
<th>PTE 511</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

- not present

---

**L1 Cache**
Page Table

- PTE 0
- PTE 1
- ...
- PTE \#PTI
- ...
- PTE 511

L1 lookup with virtual address

L1 Cache

not present
• KAISER/KPTI/KVA does not help
Foreshadow-NG Fix

- KAISER/KPTI/KVA does not help
- Only software workarounds
• KAISER/KPTI/KVA does not help
• Only software workarounds
  → Flush L1 on VM entry
• KAISER/KPTI/KVA does not help
• Only software workarounds
  → Flush L1 on VM entry
  → Disable HyperThreading
Foreshadow-NG Fix

- KAISER/KPTI/KVA does not help
- Only software workarounds
  - Flush L1 on VM entry
  - Disable HyperThreading
- Workarounds might not be complete
Meltdown Variants

A Systematic Evaluation of Transient Execution Attacks and Defenses.
Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss.
USENIX Security’19
operation \#n
Meltdown Root Cause

operation \#n

data

time
Meltdown Root Cause

operation \#n

data

data dependency

operation \#n+2

time

Michael Schwarz — www.iaik.tugraz.at
Meltdown Root Cause

operation \#n

exception

data

data dependency

operation \#n+2

transient execution

possibly architectural

time

Michael Schwarz — www.iaik.tugraz.at
Meltdown Root Cause

operation \( #n \)

data dependency

operation \( #n+2 \)

transient execution

data

possibly architectural

exception

retire

time

Michael Schwarz — www.iaik.tugraz.at
Meltdown Root Cause

Operation \( n \)

Data

Exception

Meltdown

Data dependency

Operation \( n+2 \)

Possibly architectural

Transient execution

Time
Meltdown Root Cause

operation \#n

exception
raise

Meltdown

data dependency

transient execution

operation \#n+2

data

possibly architectural

time
YOU GET A FAULT

AND YOU GET A FAULT. EVERYONE GETS A FAULT
A Systematic Evaluation of Transient Execution Attacks and Defenses.
Claudio Canella, Jo Van Bulck, Michael Schwarz, Moritz Lipp, Benjamin von Berg, Philipp Ortner, Frank Piessens, Dmitry Evtyushkin, Daniel Gruss.
USENIX Security’19
Latest Meltdown Variant: ZombieLoad

-Leaks from the fill buffer

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS’19
Latest Meltdown Variant: ZombieLoad

-Leaks from the fill buffer
-Crosses all privilege boundaries (Kernel, VM, SGX)

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS’19
Latest Meltdown Variant: ZombieLoad

- Leaks from the fill buffer
- Crosses all privilege boundaries (Kernel, VM, SGX)
- Explored microcode assists as new type of faults

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS’19
Latest Meltdown Variant: ZombieLoad

-Leaks from the **fill buffer**
- Crosses all privilege boundaries (Kernel, VM, SGX)
- Explored microcode assists as new type of faults
- Disadvantage: **minimal control** over leaked data

ZombieLoad: Cross-Privilege-Boundary Data Sampling.
Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, Daniel Gruss. CCS’19
Meltdown Outlook

- Meltdown is not a fully solved issue
• Meltdown is not a fully solved issue
• The tree is extensible
Meltdown Outlook

- Meltdown is not a fully solved issue
- The tree is extensible
- More Meltdown-type issues to come
• Meltdown is not a fully solved issue
• The tree is extensible
• More Meltdown-type issues to come
• Silicon fixes might not be complete
• Meltdown not the only transient execution attacks
• Meltdown not the only transient execution attacks
• Spectre is a second class of transient execution attacks
• Meltdown not the only transient execution attacks
• Spectre is a second class of transient execution attacks
• Instead of faults, exploit control (or data) flow predictions
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
CPU tries to predict the future (branch predictor), ... based on events learned in the past
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P’19
Speculative Execution

- CPU tries to predict the future (branch predictor), …
  - … based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, …
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ... based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ... very fast

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P’19
Speculative Execution

- CPU tries to predict the future (branch predictor), ...
  - ...based on events learned in the past
- Speculative execution of instructions
- If the prediction was correct, ...
  - ...very fast
  - otherwise: Discard results

Spectre Attacks: Exploiting Speculative Execution.
Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, Yuval Yarom. S&P’19
if (index < 4) {
  glyph[data[index]]
}
index = 0

if (index < 4) then
    glyph[data[index]]
else
    Speculate

Shared Memory

A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z

Memory

D A T A
E A R K
V E Y
...
index = 0

\[
\text{if (index < 4)} \quad \text{then}
\]

\[
glyph[data[index]]
\]

\[
\text{else}
\]

\[
\text{else}
\]

\[
\{ \text{data[0]} \}
\]

\[
\{ \text{data[1]} \}
\]

\[
\{ \text{data[2]} \}
\]

\[
\{ \text{data[3]} \}
\]

\[
\{ \cdots \}
\]
Spectre-PHT (aka Spectre Variant 1)

index = 0

Shared Memory:

A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z

Memory:

D
DATA
TAKE
KEY...

Execute:

if (index < 4) then

glyph[data[index]]

else


Spectre-PHT (aka Spectre Variant 1)

```plaintext
index = 0
if (index < 4)
    glyph[data[index]]
else
    {}
```
index = 1

if (index < 4)
then
  glyph[data[index]]
else
{}
index = 1

if (index < 4)
  then
    glyph[data[index]]
  else
    }

Shared Memory
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Memory
index = 1

if (index < 4)
  glyph[data[index]]
else
  {}
index = 1

if (index < 4) {
  glyph[data[index]]
} else {

}
index = 1

if (index < 4) then

glyph[data[index]]

else


index = 1

if (index < 4) then
  glyph[data[index]]
else
  
Memory
  data[0]
data[1]
data[2]
data[3]

Shared Memory

A B
C D E
F G H
I J K
L M N
O P Q
R S T
U V W
X Y Z
Spectre-PHT (aka Spectre Variant 1)

\[
\text{index} = 2
\]

**Shared Memory**

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

**Memory**

```
```

**Speculate**

\[
\text{glyph}[\text{data}[\text{index}]]
\]

\[
\text{if} \ (\text{index} < 4)\]

\[
\text{then}
\]

\[
\text{else}
\]

\[
\{
\}
\]
index = 2

if (index < 4) {
    glyph[data[index]]
} else {
    \{ 
    data[0]
    data[1]
    data[2]
    data[3]
    \}
Spectre-PHT (aka Spectre Variant 1)

**index = 2**

- **Shared Memory**
  - ABC
  - CDE
  - FGH
  - IJK
  - LMN
  - OPQ
  - RST
  - UVW
  - XYZ

- **Memory**
  - D
  - ATA
  - KEY
  - ... 

**Speculate**

```plaintext
if (index < 4)
    glyph[data[index]]
else
    {} 
```
index = 2

```plaintext
if (index < 4) {
    glyph[data[index]]
} else {
    T
}
```
index = 2

if (index < 4)
    glyph[data[index]]
else
    {}
index = 3

if (index < 4)
  glyph[data[index]]
else
  {}

Shared Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>D</th>
<th>data[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>data[1]</td>
</tr>
<tr>
<td>A</td>
<td>data[2]</td>
</tr>
<tr>
<td>K</td>
<td>data[3]</td>
</tr>
</tbody>
</table>

Michael Schwarz — www.iaik.tugraz.at
index = 3

if (index < 4) then
  glyph[data[index]]
else
  \{
  \}

Memory:
- data[0]
- data[1]
- data[2]
- data[3]

Shared Memory:
- A B
- C D E
- F G H
- I J K
- L M N
- O P Q
- R S T
- U V W
- X Y Z

Speculate

Michael Schwarz — www.iaik.tugraz.at
**Spectre-PHT (aka Spectre Variant 1)**

Shared Memory

```
  index = 3
```

Memory

```
if (index < 4) {
  glyph[data[index]]
} else {
}
```

```
data[0]
data[1]
data[2]
data[3]
```

```
D
A
T
A
K
E
Y
...`

---

Michael Schwarz — www.iaik.tugraz.at
index = 3

if (index < 4)

then

glyph[data[index]]

else

{}
index = 3

if (index < 4) then
  glyph[data[index]]
else
  }

Shared Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>G</td>
<td>H</td>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
<td>M</td>
<td>N</td>
<td>O</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
<td>W</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Memory

<table>
<thead>
<tr>
<th>D</th>
<th>data[0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>data[1]</td>
</tr>
<tr>
<td>A</td>
<td>data[2]</td>
</tr>
<tr>
<td>K</td>
<td>data[3]</td>
</tr>
<tr>
<td>E</td>
<td></td>
</tr>
</tbody>
</table>

Michael Schwarz — www.iaik.tugraz.at
Spectre-PHT (aka Spectre Variant 1)

Shared Memory

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>G</td>
<td>H</td>
</tr>
<tr>
<td>I</td>
<td>J</td>
</tr>
<tr>
<td>K</td>
<td>L</td>
</tr>
<tr>
<td>M</td>
<td>N</td>
</tr>
<tr>
<td>O</td>
<td>P</td>
</tr>
<tr>
<td>Q</td>
<td>R</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>U</td>
<td>V</td>
</tr>
<tr>
<td>W</td>
<td>X</td>
</tr>
<tr>
<td>Y</td>
<td>Z</td>
</tr>
</tbody>
</table>

Memory

- data[0]
- data[1]
- data[2]
- data[3]

if (index < 4) then

glyph[data[index]]

else


index = 4

if (index < 4)
    then
        glyph[data[index]]
    else
        {}
Spectre-PHT (aka Spectre Variant 1)

```
index = 4

if (index < 4)
   glyph[data[index]]
else
```

Shared Memory

```
A B C D E F G H I J K
L M N O P Q R S T U V W X Y Z
```

Memory

```
data[0]
data[1]
data[2]
data[3]
```

Speculate 

```
K
```
Spectre-PHT (aka Spectre Variant 1)

Shared Memory:

```plaintext
index = 4
```

Speculate:

```plaintext
if (index < 4)
then

glyph[data[index]]

else

{ }
```

Memory:

```plaintext
data[0]
data[1]
data[2]
data[3]
```

Michael Schwarz — www.iaik.tugraz.at
index = 4

if (index < 4)
{
    glyph[data[index]]
}

else
{
    }

Execute
operation #n

time
operation \#n

prediction

time
Spectre Root Cause

operation #n

prediction

predict CF/DF

operation #n+2

time
Spectre Root Cause

operation \#n

prediction

operation \#n+2

possibly architectural

transient execution

time

cf/df

detect
Spectre Root Cause

operation \#n

prediction

operation \#n+2

possibly architectural transient execution

predict CF/DF

time
Spectre Root Cause

- operation \( \#n \)
- prediction
- predict CF/DF
- possibly architectural
- transient execution
- operation \( \#n+2 \)
- flush pipeline on wrong prediction

Michael Schwarz — www.iaik.tugraz.at
Spectre Root Cause

- Operation \( \#n \) retire
- Prediction
  - Predict CF/DF
  - Possibly architectural
  - Transient execution

- Operation \( \#n+2 \) retire

- Flush pipeline on wrong prediction

Time
Many predictors in modern CPUs
Spectre Root Cause

- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
Spectre Root Cause

- Many predictors in modern CPUs
  - Branch taken/not taken (PHT)
  - Call/Jump destination (BTB)
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
  • Function return destination (RSB)
Many predictors in modern CPUs
- Branch taken/not taken (PHT)
- Call/Jump destination (BTB)
- Function return destination (RSB)
- Load matches previous store (STL)
• Many predictors in modern CPUs
  • Branch taken/not taken (PHT)
  • Call/Jump destination (BTB)
  • Function return destination (RSB)
  • Load matches previous store (STL)
• Most are even shared among processes
Spectre Mistraining

same address space/
in place

Victim

Victim
branch
Spectre Mistraining

Victim

Congruent branch

Address collision

Victim branch

same address space/out of place

same address space/in place
Spectre Mistraining

same address space/
out of place

same address space/
in place

Victim

Congruent branch

Address collision

Victim branch

Shared Branch Prediction State
Spectre Mistraining

same address space/ out of place

same address space/ in place

Victim

Congruent branch

Attacker

Address collision

Victim branch

Shared Branch Prediction State
Spectre Mistraining

- same address space/out of place
- same address space/in place
Spectre Mistraining

Victim

Attacker

same address space/
out of place

Congruent
branch

Congruent
branch

cross address space/
out of place

same address space/
in place

Address
collision

Address
collision

Victim
branch

Shadow
branch

Shared Branch Prediction State

Michael Schwarz — www.iaik.tugraz.at
Transient cause?
Spectre Variants

Transient cause?

Spectre-type

prediction

Transient
cause?
Spectre Variants

Transient cause?

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

microarchitectural buffer
Spectre Variants

Transient cause?

- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

Microarchitectural buffer

Mistraining strategy

Cross-address-space
Same-address-space

Cross-address-space
Same-address-space

Cross-address-space
Same-address-space

Cross-address-space
Same-address-space
Spectre Variants

Transient cause?

Spectre-type
- Spectre-PHT
- Spectre-BTB
- Spectre-RSB
- Spectre-STL

microarchitectural buffer

in-place (IP) vs., out-of-place (OP)

mistraining strategy

Cross-address-space

Same-address-space

PHT-CA-IP
PHT-CA-OP
PHT-SA-IP
PHT-SA-OP
BTB-CA-IP
BTB-CA-OP
BTB-SA-IP
BTB-SA-OP
RSB-CA-IP
RSB-CA-OP
RSB-SA-IP
RSB-SA-OP

in-place (IP) vs., out-of-place (OP)

Cross-address-space

Same-address-space

in-place (IP) vs., out-of-place (OP)
• Spectre is not a bug
• Spectre is not a bug
• It is an useful optimization
Spectre Fix

- Spectre is not a bug
- It is an useful optimization
  → Cannot simply fix it (as with Meltdown)
• Spectre is not a bug
• It is an useful optimization
→ Cannot simply fix it (as with Meltdown)
• Workarounds for critical code parts
Spectre defenses in 3 categories:

C1 Mitigating or reducing the accuracy of covert channels

C2 Mitigating or aborting speculation

C3 Ensuring secret data cannot be reached
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Defense</th>
<th>Spectre-PHT</th>
<th>Spectre-BTB</th>
<th>Spectre-RSB</th>
<th>Spectre-STL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attack</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (〇), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (□), or out of scope (◇).
### Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Retpoline</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD / SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
- Attack is mitigated (●), partially mitigated (○), not mitigated (), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◊).
### Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spectre-PHT</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

- | | | | |
| | | | | |
| | | | | |
| | | | | |

Attack is mitigated (●), partially mitigated (○), not mitigated (○), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (□), or out of scope (◇).
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB</th>
<th>Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD / SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-PHT</td>
<td>○</td>
<td></td>
<td></td>
<td>○</td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○</td>
<td>○</td>
<td>○</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (☐), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (☐), or out of scope (◇).
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spectre-PHT</td>
</tr>
<tr>
<td></td>
<td>Spectre-BTB</td>
</tr>
<tr>
<td></td>
<td>Spectre-RSB</td>
</tr>
<tr>
<td></td>
<td>Spectre-STL</td>
</tr>
<tr>
<td>Intel</td>
<td></td>
</tr>
<tr>
<td></td>
<td>InvisSpec</td>
</tr>
<tr>
<td></td>
<td>SafeSpec</td>
</tr>
<tr>
<td></td>
<td>DAWG</td>
</tr>
<tr>
<td></td>
<td>RSB Stuffing</td>
</tr>
<tr>
<td></td>
<td>Retpoline</td>
</tr>
<tr>
<td></td>
<td>Poison Value</td>
</tr>
<tr>
<td></td>
<td>Index Masking</td>
</tr>
<tr>
<td></td>
<td>Site Isolation</td>
</tr>
<tr>
<td></td>
<td>SLH</td>
</tr>
<tr>
<td></td>
<td>YSNB</td>
</tr>
<tr>
<td></td>
<td>IBRS</td>
</tr>
<tr>
<td></td>
<td>STIPB</td>
</tr>
<tr>
<td></td>
<td>IBPB</td>
</tr>
<tr>
<td></td>
<td>Serialization</td>
</tr>
<tr>
<td></td>
<td>Taint Tracking</td>
</tr>
<tr>
<td></td>
<td>Timer Reduction</td>
</tr>
<tr>
<td></td>
<td>Sloth</td>
</tr>
<tr>
<td></td>
<td>SSBD/SSBB</td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (◯), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (◇), or out of scope (◇).
<table>
<thead>
<tr>
<th>Attack</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

Attack is mitigated (●), partially mitigated (○), not mitigated (☐), theoretically mitigated (■), theoretically impeded (□), not theoretically impeded (☐), or out of scope (◇).
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD</th>
<th>SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td></td>
<td>□</td>
<td>□</td>
<td>□</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spectre-STL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

> Attack is mitigated (●), partially mitigated (○), not mitigated (□), theoretically mitigated (■), theoretically impeded (■), not theoretically impeded (□), or out of scope (◇).
## Spectre: Defense Analysis

<table>
<thead>
<tr>
<th>Attack</th>
<th>Defense</th>
<th>InvisiSpec</th>
<th>SafeSpec</th>
<th>DAWG</th>
<th>RSB</th>
<th>Stuffing</th>
<th>Poison Value</th>
<th>Index Masking</th>
<th>Site Isolation</th>
<th>SLH</th>
<th>YSNB</th>
<th>IBRS</th>
<th>STIPB</th>
<th>IBPB</th>
<th>Serialization</th>
<th>Taint Tracking</th>
<th>Timer Reduction</th>
<th>Sloth</th>
<th>SSBD/SSBB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spectre-PHT</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>☘</td>
<td>☘</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>Spectre-BTB</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>☘</td>
<td>☘</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>Spectre-RSB</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>☺</td>
<td>☺</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
</tr>
<tr>
<td>Spectre-STL</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
<td>☘</td>
</tr>
</tbody>
</table>

Attack is mitigated (☉), partially mitigated (☉), not mitigated (☉), theoretically mitigated (◼), theoretically impeded (□), not theoretically impeded (□), or out of scope (◇).
• Many countermeasures only consider the cache to get data...
• Many countermeasures **only consider the cache** to get data...
• ...but there are other possibilities, e.g.,
Many countermeasures only consider the cache to get data...
...but there are other possibilities, e.g.,
  - Port contention (SMoTherSpectre)
Many countermeasures only consider the cache to get data...
...but there are other possibilities, e.g.,
- Port contention (SMoTherSpectre)
- AVX (NetSpectre)
• Many countermeasures only consider the cache to get data...
• ...but there are other possibilities, e.g.,
  • Port contention (SMoTherSpectre)
  • AVX (NetSpectre)
  • TLB (Store-to-Leak Forwarding)
Many countermeasures only consider the cache to get data...
...but there are other possibilities, e.g.,
- Port contention (SMoTherSpectre)
- AVX (NetSpectre)
- TLB (Store-to-Leak Forwarding)

Cache is just the easiest
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped
Written by Michael Larabel in Linux Kernel on 24 November 2018 at 09:00 AM EST. 6 Comments

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Linux 4.19.4 & 4.14.83 Released With STIBP Code Dropped

On Friday marked the release of the Linux 4.19.4 kernel as well as 4.14.83 and 4.9.139.

Greg Kroah-Hartman issued this latest round of stable point releases as basic maintenance updates. While these point releases don't tend to be too notable and generally go unmentioned on Phoronix, this round is worth pointing out since 4.19.4 and 4.14.83 are the releases that end up reverting the STIBP behavior that applied Single Thread Indirect Branch Predictors to all processes on supported systems. That is what was introduced in Linux 4.20 and then back-ported to the 4.19/4.14 LTS branches, which in turn hurt the performance a lot. So for now the code is removed.

As covered yesterday, there is improved STIBP code on the way for Linux 4.20 that by default just apply STIBP to SECCOMP threads and processes requesting it via prctl() but otherwise is off by default (that behavior can also be changed via kernel parameters).
Current mitigations are either incomplete or cost performance.
- Current mitigations are either **incomplete** or cost performance

→ More **research** required
• Current mitigations are either incomplete or cost performance
  → More research required
• Both on attacks and defenses
• Current mitigations are either incomplete or cost performance
→ More research required
• Both on attacks and defenses
→ Efficient defenses only possible when attacks are known
Leaking Data

- Side channels so far
  - leak meta data
Leaking Data

- Side channels so far
  - leak meta data
  - covertly transmit data
Leaking Data

- Side channels so far
  - leak meta data
  - covertly transmit data

- As a building block
  - leak data
Leaking Data

- Side channels so far
  - leak meta data
  - covertly transmit data
- As a building block
  - leak data
- What about modifying data?
DRAM organization

channel 0
back of DIMM: rank 1
front of DIMM: rank 0
DRAM organization

channel 0

channel 1

back of DIMM: rank 1
front of DIMM: rank 0
DRAM organization

channel 0

channel 1

back of DIMM: rank 1

front of DIMM: rank 0
DRAM organization

- Channel 0
  - Front of DIMM: rank 0
  - Back of DIMM: rank 1

- Channel 1
  - Chip

Michael Schwarz — www.iaik.tugraz.at
DRAM organization

bank 0

row 0
row 1
row 2
...
row 32767

row buffer
DRAM organization

chip

bank 0

row 0
row 1
row 2
...
row 32767

row buffer

64k cells
1 capacitor, 1 transistor each
How reading from DRAM works

CPU wants to access row 1
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer
How reading from DRAM works

DRAM bank

CPU wants to access row 2

row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2

→ row 2 activated

→ row 2 copied to row buffer
How reading from DRAM works

CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
→ slow (row conflict)
How reading from DRAM works

CPU wants to access row 2—again
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
How reading from DRAM works

CPU wants to access row 2—again
→ row 2 already in row buffer
→ fast (row hit)
How reading from DRAM works

row buffer = cache
DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks.
Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, Stefan Mangard. USENIX Security’16
Rowhammer

DRAM bank

111111111111111
111111111111111
111111111111111
111111111111111
... 
111111111111111

row buffer

Cells leak faster upon proximate accesses → Rowhammer
Cells leak faster upon proximate accesses → Rowhammer
Cells leak faster upon proximate accesses → Rowhammer
Cells leak faster upon proximate accesses → Rowhammer
Cells leak faster upon proximate accesses → Rowhammer
Rowhammer

Cells leak faster upon proximate accesses → Rowhammer
How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)
How widespread is the issue?

- 85% affected (estimation 2014)
- 52% affected (estimation 2015)

First believed to be safe
- We showed bit flips in 2016
- 67% affected (estimation 2016)
• Single bit flips allow
  • modifying instructions
• Single bit flips allow
  • modifying instructions
  • breaking cryptography
• Single bit flips allow
  • modifying instructions
  • breaking cryptography
  • changing permissions
Single bit flips allow:
- modifying instructions
- breaking cryptography
- changing permissions
- crashing systems
Single bit flips allow

- modifying instructions
- breaking cryptography
- changing permissions
- crashing systems
- ...

In software, no permissions required
Single bit flips allow:
- modifying instructions
- breaking cryptography
- changing permissions
- crashing systems
- ...

In software, no permissions required.
Future

- More attacks exploiting performance optimizations in hardware
  - New variants are disclosed frequently
Microarchitectural Data Sampling (MDS)
Transient Execution Attacks

Transient cause:

Spectre-type

- Spectre-PHT
  - Cross-address-space
    - PHT-CA-IP
    - PHT-CA-OP
  - Same-address-space
    - PHT-SA-IP
    - PHT-SA-OP

- Spectre-BTB
  - Cross-address-space
    - BTB-CA-IP
    - BTB-CA-OP
  - Same-address-space
    - BTB-SA-IP
    - BTB-SA-OP

- Spectre-RSB
  - Cross-address-space
    - RSB-CA-IP
    - RSB-CA-OP
  - Same-address-space
    - RSB-SA-IP
    - RSB-SA-OP

- Spectre-STL
  - Cross-address-space
    - STL-CA-IP
    - STL-CA-OP
  - Same-address-space
    - STL-SA-IP
    - STL-SA-OP

Meltdown-type

- Meltdown-US
  - Cross-address-space
    - US-CA-IP
    - US-CA-OP
  - Same-address-space
    - US-SA-IP
    - US-SA-OP

- Meltdown-NM-REG
- Meltdown-PF
- Meltdown-BR
- Meltdown-GP
- Meltdown-MCA
- Meltdown-MPX
- Meltdown-BND
- Meltdown-CPL-REG
- Meltdown-NC-REG
- Meltdown-AD
- Meltdown-AVX-LP
- Meltdown-US-L1
- Meltdown-US-LFB
- Meltdown-US-SB
- Meltdown-P-L1
- Meltdown-P-LFB
- Meltdown-P-SB
- Meltdown-P-LP
- Meltdown-AD-LFB
- Meltdown-AD-SB
• Transient Execution Attacks are...
Transient Execution Attacks are...
  ...a novel class of attacks
Transient Execution Attacks

- Transient Execution Attacks are...
  - ...a novel class of attacks
  - ...extremely powerful
Transient Execution Attacks are...

- ...a novel class of attacks
- ...extremely powerful
- ...only at the beginning
Transient Execution Attacks are...
- ...a novel class of attacks
- ...extremely powerful
- ...only at the beginning

- Many optimizations introduce side channels → now exploitable
A unique chance to

- rethink processor design
- grow up, like other fields (car industry, construction industry)
• Optimizations in hardware often lead to side channels
• **Optimizations** in hardware often lead to side channels
• Unknown and **novel** side channels are likely to exist
• Optimizations in hardware often lead to side channels
• Unknown and novel side channels are likely to exist
• Next to no permissions required for attacks
• Optimizations in hardware often lead to side channels
• Unknown and novel side channels are likely to exist
• Next to no permissions required for attacks
• Building countermeasures is extremely hard
BRACE YOURSELVES
MORE BUGS ARE COMING
Any Questions?