SpectreRewind: Leaking Secrets to Past Instructions

Jacob Fustos, Michael Bechtel, Heechul Yun
University of Kansas, USA
Speculative Execution Attacks

- Attacks exploiting microarchitectural side-effects left by speculative (transient) instructions
- Many variants: Spectre, Meltdown, Foreshadow, MDS, LVI, ...
- Secrets are transferred over microarchitectural covert channels
- Most known attacks use cache covert channels
Cache-based Covert Channels

• Exploit that speculatively executed memory instructions change cache
• Encode secret by executing secret dependent memory accesses
• Use cache hit/miss timing differences to recover the secret
  • Cache miss $\rightarrow$ 0 (was not accessed)
  • Cache hit $\rightarrow$ 1 (was accessed)
• Secret is recovered after transient executions are squashed

![Diagram of Traditional Spectre Attack]

3
Cache Covert Channel Mitigation Solutions

- Prevent speculative execution from modifying the cache
  - Buffer speculative results in additional “shadow” hardware structures
- Undo cache changes. E.g., CleanupSpec [Saileshwar et al., MICRO’19]

InvisiSpec [Yan et al., MICRO’18]

SafeSpec [Khasawneh et al., DAC’19]
Contestation-based Covert Channels on SMT

- Exploit that contention on shared functional units/ports between Simultaneous multithreading (SMT) hardware threads
- Secret is transmitted **during** the speculative execution
  - Bypass stateful covert channel defenses such as InvisiSpec
- Can be mitigated by
  - Disabling SMT
  - Preventing co-scheduling of different domains

![Contention channel on SMT Core](image)

*smotherSpectre [Bhattacharyya+, CCS 19]*
Outline

• Background
• **SpectreRewind**
• Evaluation
• Conclusion
SpectreRewind

• A novel approach for creating contention-based covert channels
• From a single hardware thread (doesn’t require SMT)
• Transmit secret to past instructions
• Bypasses existing cache covert channel defenses

• Key insight:
  • Contention-based covert channels w/o SMT are possible
  • Contention between speculative and past non-speculative instructions can create covert channels
Our Approach

• Receiver: instructions executed *before* speculative execution
• Sender: instructions conditionally (secret dependent) executed *during* speculative execution
• Both sender and receiver are in the same hardware thread
• Contention between the sender and the receiver creates a channel
  • 0 – fast
  • 1 – slow (due to contention)
• Target *non-pipelined functional units*

![SpectreRewind Covert Channel](image)
## Pipelined vs. Non-pipelined Function Units

<table>
<thead>
<tr>
<th>Fully pipelined</th>
<th>Non-pipelined</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Receiver</strong></td>
<td><strong>Sender</strong></td>
</tr>
<tr>
<td><img src="image" alt="Scheduler" /></td>
<td><img src="image" alt="Scheduler" /></td>
</tr>
<tr>
<td><img src="image" alt="Ready" /></td>
<td><img src="image" alt="Ready" /></td>
</tr>
<tr>
<td><img src="image" alt="Waiting" /></td>
<td><img src="image" alt="Waiting" /></td>
</tr>
<tr>
<td><img src="image" alt="Integer Multiplier" /></td>
<td><img src="image" alt="Integer Multiplier" /></td>
</tr>
<tr>
<td><img src="image" alt="Stage 1" /></td>
<td><img src="image" alt="Stage 1" /></td>
</tr>
<tr>
<td><img src="image" alt="Stage 2" /></td>
<td><img src="image" alt="Stage 2" /></td>
</tr>
<tr>
<td><img src="image" alt="Stage 3" /></td>
<td><img src="image" alt="Stage 3" /></td>
</tr>
</tbody>
</table>

(a) Fully pipelined
- No impact to receiver execution time

(b) Non-pipelined
- Receiver execution time is prolonged
Floating Point Division Covert Channel

• Start a timer
• Perform multiple divisions
• Cause a mis-speculation
• Calculate a bit to transmit
• If bit is ‘1’ do more division
• Cause contention with receiver
• Time entire attack
Outline

• Background
• SpectreRewind
• **Evaluation**
• Conclusion
Evaluation Platforms and Methodology

- Used eight platforms from Intel, AMD, and ARM
- Native SpectreRewind PoC
  - Written in standard C language
- Timing measurement methods
  - RDTSC instructions on Intel and AMD (x86-64)
  - A counting thread on ARM (arm64)
- Methodology
  - Send 1 million known ‘0’ and ‘1’ bit values over the division channel and measure the timings
Channel Properties

• Clearly distinguishable patterns on all tested platforms
Performance Analysis

- High transfer rates and low error rates

<table>
<thead>
<tr>
<th>CPU</th>
<th>Microarch.</th>
<th>Latency (cycles)</th>
<th>Throughput (cycles)</th>
<th>Transfer Rate (KB/s)</th>
<th>Error Rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i5-8250U</td>
<td>Kabylake R</td>
<td>13–15</td>
<td>4</td>
<td>53.1</td>
<td>0.02</td>
</tr>
<tr>
<td>Intel Core i5-6500</td>
<td>Skylake</td>
<td>13–15</td>
<td>4</td>
<td>105.3</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Intel Core i5-6200U</td>
<td>Skylake</td>
<td>13–15</td>
<td>4</td>
<td>74.9</td>
<td>0.04</td>
</tr>
<tr>
<td>Intel Xeon E5-2658 v3</td>
<td>Haswell</td>
<td>10–20</td>
<td>8</td>
<td>64.1</td>
<td>&lt;0.01</td>
</tr>
<tr>
<td>Intel Core i5-3340M</td>
<td>Ivybridge</td>
<td>10–20</td>
<td>8</td>
<td>75.6</td>
<td>0.16</td>
</tr>
<tr>
<td>AMD Ryzen 3 2200G</td>
<td>Zen</td>
<td>8–13</td>
<td>4</td>
<td>83.1</td>
<td>5.50</td>
</tr>
<tr>
<td>AMD Ryzen 5 2600</td>
<td>Zen+</td>
<td>8–13</td>
<td>4</td>
<td>84.8</td>
<td>3.30</td>
</tr>
<tr>
<td>NVIDIA Jetson Nano</td>
<td>Cortex A57</td>
<td>N/A</td>
<td>N/A</td>
<td>87.7</td>
<td>0.02</td>
</tr>
</tbody>
</table>
Google Chrome Sandbox

• Implemented a SpectreRewind PoC in JavaScript on Chrome
• Noisier but still distinguishable timing differences
Outline

• Background
• SpectreRewind
• Evaluation
• Conclusion
Discussion

• Benefits
  • Does not require SMT hardware (single thread)
  • Defeats all known hardware solutions for stateful (cache) covert channels
  • Alternative to cache-based covert channels like Flush+Reload

• Limitations
  • Limited to same address space attacks
  • Finding division-based gadgets may be difficult
  • Attacker controls both receiver and sender
Conclusion

• A novel approach for creating contention-based covert channels
  • Transmit secret to *past instructions*
  • Bypass existing cache covert channel defenses

• Exploits contention on *non-pipelined functional units*
  • Between speculative and past non-speculative instructions
  • From a single hardware thread

• Floating point division unit based covert channel
  • ~100KB/s transfer rate, <1% error
  • Works across CPUs from Intel, AMD, and ARM