Fast, Robust and Accurate Detection of Cache-based Spectre Attack Phases

Arash Pashrashid, Ali Hajiabadi, Trevor E. Carlson

National University of Singapore
Breaking News: Fundamental Security Problem of Modern CPUs

• Computer architects’ main focus was on CPU performance for decades
• However, modern CPUs can leak sensitive data like passwords, cryptographic keys via unexpected side channels
• Affects all modern processors, servers, smart phones
  • Intel, AMD, ARM, IBM, etc.
  • Affects all operating systems

Recipe for Spectre Attack

• Speculative Execution + Side-Channels
  • Speculative Execution:
  • Side-Channels:
    • Attacker analyzes these channels to extract victim’s secret dependent activities
    • For example, last level cache is one of the common side-channels
Example: Speculative Execution Attack via Cache

1. Initialization
   - Branch_mistraining()
   - prime_μarch_state()
   - if (safety check)
   - secret = load(...)
   - side-channel(secret)
   - probe_μarch_state()

2. Victim Execution
   - Not secret
   - Not secret
   - Not secret
   - Secret
   - Victim
   - Attacker
   - Attacker
   - Attacker
   - Attacker

3. Probe
   - Initiates the system for attack
   - Misspeculation
   - Load Secret
   - Secret-dependent trace in the cache
   - Recovering the Secret
   - e.g., access specific cache lines based on the secret value
   - e.g., load latency
Why Using Side-Channel Attacks Detectors?

- **Problem:** Comprehensive Spectre mitigation incurs significant performance overhead (Up to 2x)\(^1,2\)
- Not all these overheads are necessary to provide the secure system
- One possible solution: If not attacks are present, no expensive mitigations needed
- **Our goal:** Addressing the limitations of existing detectors

---

Insight: Is Machine Learning the Solution?

• The ideal detector should be **fast, accurate, robust** and **efficient**
• Machine learning is widely deployed for the SCA detectors\(^1,2,3\)

**Question:** Are ML-based methods robust to evasive attacks or benign applications?

• In this work, we propose the evasive attacks to break ML-based detectors

---

Evasive Spectre Attacks

• **Goal:** To change the footprint of a Spectre attack without compromising the attack’s success

• **How:** Expanding the branch mistraining without disrupting the attack’s success

- Two variants of this attack:
  - insertion of NOPs to the branch mistraining part of the original Spectre
  - insertion of memory delay instructions

- Accuracy of PerSpectron, the SOTA ML-Based detector, drops from 99% to 14%
Evasive Spectre Attacks

- **Goal:** Performing all the essential steps of attack from benign programs

- **How:** Finding similar behavior inside benign programs for each step
  1. Branch mistraining (Attacker): A loop with a large number of iterations.
  2. Side-channel initialization phase (Attacker): Initialization of a large array
  3. Secret recovering phase (Attacker): Same with Phase 2
  4. By linking the selected slices that represent each step, a full attack can be launched

- Accuracy of PerSpectron drops from 99% to 12%
Motivation

• Limitations of state-of-the-art ML-based detectors:
  • They are fragile to our Expanded-Spectre
  • Also, they can befooled by our Benign-Program-Spectre

• We need to design an SCA detector to overcome these shortcomings
  • To be robust to our evasive Spectre attacks
  • And to be accurate, fast, and efficient

• We design Spectify to get closer to an ideal detector
Spectify Detection Methodology

- We aim to track the sequence of attack phases
- Using a direct-analysis approach to monitor microarchitectural state changes
- **Init transition**: If enough number of cache lines are initialized
- **Spec transition**: If a sufficient number of cache lines are initialized by previous processes and the current process speculatively accesses one of initialized cache lines
- **Squash transition**: If misprediction happens and the state of only one of the initialized cache lines is changed
1. Modifying the ROB to track unresolved branches

2. Adding new tables (PAST and FAST) to track the accessed/flushed locations

3. If misprediction occurs, then track which locations accessed during speculation window

4. If context-switch occurs, then the new tables checked for a data leak. Also, taking a checkpoint before CS in a new table (PAST-Hist)
Experimental Setup

• Simulation:
  • `gem5` in syscall emulation mode
  • CACTI 6.5 for power and area overheads

• Benchmarks:
  • **Benign programs:** SPEC CPU2006 benchmark suite
  • **Malicious programs:** Spectre V1, Spectre V2, different cache attacks, and our evasive Spectre attacks
  • **Representatives:** ELFies as executable representative with a region size of 100M instructions

• PerSpectron Experimental Setup
  • FANN C library for the implementation of neural networks
  • 10k instruction sampling rate
  • Single-layer perceptron neural network
  • 66% of the data is used for training and the rest for testing
Comparison of PerSpectron and Spectify

• Both PerSpectron and Spectify show high accuracy for Benign, Spectre V1 and V2

• While PerSpectron accuracy falls from 99% to 14% for Expanded-Spectre attack, there is no accuracy reduction for Spectify

• While PerSpectron accuracy falls from 99% to 12% for Benign-Program-Spectre attack, there is no accuracy reduction for Spectify

• Even retraining PerSpectron with our evasive Spectre doesn’t give acceptable accuracy to the PerSpectron

• Even the false positive rate in Spectify is less than PerSpectron and is around 0.02%

<table>
<thead>
<tr>
<th>Test Scenario</th>
<th>PerSpectron</th>
<th>Spectify</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benign</td>
<td>99.10%</td>
<td>99.98%</td>
</tr>
<tr>
<td>Spectre V1</td>
<td>99.61%</td>
<td>100.0%</td>
</tr>
<tr>
<td>Spectre V2</td>
<td>98.67%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>
## Running Spectify with SPEC CPU2006

<table>
<thead>
<tr>
<th>Application</th>
<th>#frames</th>
<th>#min 2 sets primed</th>
<th>#Data Leaks</th>
</tr>
</thead>
<tbody>
<tr>
<td>401.bzip2</td>
<td>38136</td>
<td>6667</td>
<td>4</td>
</tr>
<tr>
<td>403.gcc</td>
<td>151771</td>
<td>53186</td>
<td>11</td>
</tr>
<tr>
<td>410.bwaves</td>
<td>55255</td>
<td>46278</td>
<td>3</td>
</tr>
<tr>
<td>416.gameess</td>
<td>26720</td>
<td>1205</td>
<td>10</td>
</tr>
<tr>
<td>429.mcf</td>
<td>217673</td>
<td>106053</td>
<td>52</td>
</tr>
<tr>
<td>434.zeusmp</td>
<td>32763</td>
<td>19760</td>
<td>40</td>
</tr>
<tr>
<td>436.cactusADM</td>
<td>60729</td>
<td>7407</td>
<td>0</td>
</tr>
<tr>
<td>444.namd</td>
<td>277321</td>
<td>244</td>
<td>0</td>
</tr>
<tr>
<td>445.gobmk</td>
<td>48742</td>
<td>4074</td>
<td>1</td>
</tr>
<tr>
<td>450.soplex</td>
<td>128519</td>
<td>39411</td>
<td>10</td>
</tr>
<tr>
<td>462.libquantum</td>
<td>72327</td>
<td>26315</td>
<td>0</td>
</tr>
<tr>
<td>471.omnetpp</td>
<td>85982</td>
<td>22715</td>
<td>2</td>
</tr>
</tbody>
</table>

**Number of times at least 2 sets are primed**

**Actual memory data leaks that potentially can be exploited**

- Demonstrates the possibility of initialization for Benign-Program-Spectre from the SPEC programs
- Demonstrates that our Benign-Program-Spectre is possible
Efficiency Analysis of Spectify

• No performance overhead: Operates in parallel with the main processor core, off the critical path

• Power overhead: 0.66% over the baseline core
  • Most overheads come from FAST, PAST, and PAST-Hist
  • Direct-mapped cache structures are relatively efficient

• The area overhead: 7.3% over the baseline core
Conclusion

• We break the state-of-the-art detector, PerSpectron, by our evasive Spectre
  • Expanded-Spectre
  • Benign-Program-Spectre

• We propose a new detector to satisfy ideal detector conditions
  • 100% accuracy for our tested applications ✓
  • Detection before attack completion ✓
  • Robust to our evasive Spectre attacks ✓
  • No performance overhead, 0.66% power overhead, and 7.3% area overhead ✓
Fast, Robust and Accurate Detection of Cache-based Spectre Attack Phases

Arash Pashrashid, Ali Hajiabadi, Trevor E. Carlson

National University of Singapore