SpectreGuard: An Efficient Data-centric Defense Mechanism against Spectre Attacks

Jacob Fustos, Farzad Farshchi, Heechul Yun
University of Kansas
Speculative Execution Attacks

• Attacks exploiting microarchitectural side-effects of executing speculative (transient) instructions
• Many variants

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variant 1 (Spectre) [16]</td>
<td>Bounds Check Bypass</td>
</tr>
<tr>
<td>Variant 1.1 [15]</td>
<td>Bounds Check Bypass Store</td>
</tr>
<tr>
<td>Variant 1.2 [15]</td>
<td>Read-only Protection Bypass</td>
</tr>
<tr>
<td>Variant 2 (Spectre) [16]</td>
<td>Branch Target Injection</td>
</tr>
<tr>
<td>Variant 3 (Meltdown) [18]</td>
<td>Supervisor Protection Bypass</td>
</tr>
<tr>
<td>Variant 3a [12]</td>
<td>System Register Bypass</td>
</tr>
<tr>
<td>Lazy FP [24]</td>
<td>FPU Register Bypass</td>
</tr>
<tr>
<td>Variant 4 [9]</td>
<td>Speculative Store Bypass</td>
</tr>
<tr>
<td>ret2spec [20]</td>
<td>Return Stack Buffer</td>
</tr>
<tr>
<td>L1 Terminal Fault [11, 26]</td>
<td>Virtual Translation Bypass</td>
</tr>
</tbody>
</table>

No hardware support planned in near future
Spectre Attack (Variant 1)

if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}

• Assume x is under the attacker’s control
• Attacker trains the branch predictor to predict the branch is in-bound
Spectre Attack (Variant 1)

```java
if(x < array1_length){
    val = array1[x];  // 1. [ACCESS]
    tmp = array2[val*512];
}
```

• Speculative execution of the first line **accesses** the secret (array1[x])
Spectre Attack (Variant 1)

```c
if(x < array1_length){
    val = array1[x];
    tmp = array2[val*512];
}
```

• Speculative execution of the second, secret dependent load transmits the secret to a microarchitectural state (e.g., cache)
Spectre Attack (Variant 1)

```cpp
if(x < array1_length) {
    val = array1[x];
    tmp = array2[val*512];
}
```

3. [RECEIVE]

- Attacker **receives** the secret by timing access latency differences (cache hit vs. miss) among the elements in the probe array
  - Flush+reload, prime+probe, ...

• Attacker receives the secret by timing access latency differences (cache hit vs. miss) among the elements in the probe array
Existing Software Mitigation

if(x < array1_length) {
    _mm_lfence();
    val = array1[x];
    tmp = array2[val*512];
}

• Manually stop speculation
  • By inserting ‘lfence’ instructions [Intel, 2018]
  • Or by introducing additional data dependencies [Carruth, 2018]
  • Error prone, high programming complexity, performance overhead
Existing Hardware Mitigation

• Hide speculative execution
  • By buffering speculative results into additional “shadow” hardware structures
  • High complexity, high overhead (performance, space)

InvisiSpec [Yan et al., MICRO’18]

SafeSpec [Khasawneh et al., DAC’19]
SpectreGuard

• Data-centric software/hardware collaborative approach
  • Software tells hardware what data (not code) needs protection
  • Hardware selectively protects the identified data from Spectre attacks

• Key observations
  • Not all data is secret
  • Not all speculative loads in a vulnerable code leak secret
Obs. 1: Not All Data Is Secret

- **Non-sensitive data**
  - Most program code, data
  - **Optimize for performance**

- **Sensitive (secret) data**
  - Cryptographic keys, passwords, ...
  - **Optimize for security**

**Memory**

- Attacker’s controlled data
- AES encryption table
- Other public information
- RSA private key
- Bank account information
- Other secret data
Obs. 2: Not All Speculative Loads Leak Secret

• The first load does NOT leak secret
• The second, secret dependent load leaks the secret
• Delay the secret dependent load until after the branch is resolved

```java
if(x < array1_length){
    val = array1[x]; // 1. [ACCESS]
    tmp = array2[val*512]; // 2. [TRANSMIT]
}
```

........
Approach

• Step 1: Software tells OS what data is secret
• Step 2: OS updates the page table entries
• Step 3: Load of the secret data is identified by MMU
• Step 4: Non-speculative data forwarding is delayed until safe
• **Non-Speculative (NS) memory regions**
  • Memory regions that may contain secret
  • Declared by software through a system call (`mmap`) or ELF header
  • Updated by OS in the page table (a single bit `NS` flag per page)
Gem5 Implementation
Evaluation Setup

• Full system simulation using Gem5 (O3CPU model) and Linux kernel (4.18)

<table>
<thead>
<tr>
<th>Core</th>
<th>Single-core (x86 ISA), 8 issue, out-of-order, 2 GHz IQ: 64, ROB: 192, LSQ: 32/32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>Private L1-I/D: 16/64 KiB (4/8-way), 1 cycle latency</td>
</tr>
<tr>
<td></td>
<td>Shared L2: 256 KiB (16-way), 8 cycle latency</td>
</tr>
<tr>
<td>DRAM</td>
<td>Read/write buffers: 32/64, open-adaptive policy</td>
</tr>
<tr>
<td></td>
<td>DDR3@800MHz, 1 rank, 8 banks</td>
</tr>
</tbody>
</table>

• Comparison
  • Native: unmodified baseline system
  • InvisiSpec: a fully hardware solution [Yan et al., Micro’18]
  • Fence: a fully software solution (insert lifence after all branches)
  • SG: SpectreGuard
Synthetic Workloads

- *(S)pectre*: contains Spectre gadget; does not access the secret key
- *En(C)ryption*: background communication, access the secret key
Results of Synthetic Workloads

- Varies percent time spent in S and C
- $SG(\text{Key})$ achieves native performance
  - Only secret key is marked non-speculative
- $SG(\text{All})$ achieves comparable performance with InvisiSpec
  - All memory (code, data, heap, stack) is marked non-speculative (NS)
Results of SPEC2006 Benchmarks

- **SG(All)** achieves comparable performance with *InvisiSpec*
- **SG(Heap)** achieves better performance than *InvisiSpec*
  - Only **heap** is marked as non-speculative (NS) pages
- **SpectreGuard** enables targeted security and performance trade-offs
Conclusion

• Speculative execution attacks
  • Affect all high-performance out-of-order processors
  • Existing software mitigation suffers high programming complexity/overhead
  • Hardware only mitigation is costly

• SpectreGuard
  • A data-centric software/hardware collaborative defense mechanism
  • Low programming effort (identifying secret data, not vulnerable code)
  • Low hardware cost (no additional "shadow" structure)
  • Effective, targeted defense against Spectre attacks

https://github.com/CSL-KU/SpectreGuard
Future Work

• FPGA implementation extending an open-source RISC-V SoC
• Additional compiler/library support to aid programmers
• Apply our data-centric approach to address other speculative execution attacks
Thank You!

Disclaimer:

This research is supported by NSF CNS 1718880 and NSA Science of Security initiative contract #H98230-18-D-0009.