Secure Hierarchy-Aware Cache Replacement Policy (SHARP):
Defending Against Cache-Based Side Channel Attacks

Mengjia Yan, Bhargava Gopireddy, Thomas Shull, Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

Presented by Mengjia Yan
Shared Resources in Cloud

- Shared hardware resources can leak information
- **Cache side-channel attacks**: attacker observes a victim’s cache behavior
  - Can bypass software security policies
  - Leave no trace
Cache Side-Channel Attacks are Increasing

- Public cloud → Personal devices
- Cryptography → Everyday applications

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Existing Defense Schemes

• Avoid co-residency
  → Low Resource Utilization

• Cache partition
  – Process-based partition
  – Region-based partition
  → High performance overhead
    - Require code modification
    - Difficult to precisely determine addresses to protect

• Runtime diversification
  – Add noise to timing system
  – Randomize cache mapping
  → Affect normal applications, can not defend against storage-based attack
  → High performance overhead
Attack Illustration – Evict+Reload

Victim L1 Cache

Spy L1 Cache

Cache Set

Shared L2 Cache (Inclusive)
Attack Illustration – Evict+Reload

Victim L1 Cache

Spy L1 Cache

Shared L2 Cache (Inclusive)

Probe address
Attack Illustration – Evict+Reload

Victim L1 Cache

Spy L1 Cache

Inclusion Victim

Shared L2 Cache (Inclusive)

Probe address
Spy’s line
Access
Evict
Wait
Analyze

Evict
Conflict
Hit

Time
Attack Illustration – Evict+Reload – Cont’d

- **Spy Access**
  - Miss
  - Spy's line
- **Memory Access**
- **Shared L2 Cache (Inclusive)**
- **Time**
  - Evict
  - Wait
  - Analyze
  - No Access
- **Probe address**
- **Spy L1 Cache**
- **Victim L1 Cache**

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Cache Side-Channel Attack Classification

Eviction Strategies

- Conflict-based
- Flush-based

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Contributions

- Insight: Conflict-based attacks rely on “Inclusion Victims”

- Introduce SHARP:
  - A shared cache replacement policy that defends against conflict-based attacks by preventing inclusion victims
  - A slightly modified “clflush” instruction to prevent flush-based attacks
SHARP: Preventing Inclusion Victims

**Step 1:** Find a cache line in the set that is not present in any private cache
SHARP: Preventing Inclusion Victims

Step 1: Find a cache line in the set that is not present in any private cache

Inclusion victim from other core is prevented

Victim L1 Cache

Spy L1 Cache

Shared L2 Cache

- Probe address
- Spy’s line
- Line not in any private cache
SHARP: Prevention of MultiCore Attack

Step 1: Find a cache line in the set that is not present in any private cache

Otherwise

Step 2: Find a cache line in the set that is present only in the requesting core’s private cache
SHARP: Prevention - MultiCore Attack

Step 2: Find a cache line in the set that is present only in the requesting core’s private cache
SHARP: Prevention - MultiCore Attack

Step 2: Find a cache line in the set that is present only in the requesting core’s private cache.

Inclusion victim from other core is prevented.

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
SHARP Summary

Step 1: Find a cache line in the set that is not present in any private cache

Otherwise

Step 2: Find a cache line in the set that is present only in the requesting core’s private cache

Otherwise

Step 3: Randomly evict a line, increment alarm counter

SHARP needs to know whether a line is present in private cache
- Use presence bits in directory (Core Valid Bits)
- Query, with a message, the private caches for information
Preventing Flush-Based Attacks

- “clflush” instruction
  - Invalidates an address from all levels of the cache hierarchy
  - Can be used at all privilege levels on any address

- Used to handle inconsistent memory states
  - Memory-mapped IO
  - Self modifying code

- Attacker exploits “clflush” through page sharing of

SHARP: Allow “clflush” only if the thread has write access to the address
Experimental Setup

- MarssX86 cycle-level full system simulator

- 2 to 16 out of order cores
  - Private DL1, IL1, L2 (32KB, 32KB, 256KB)
  - Shared Inclusive L3 cache (2MB slice per core)
  - Baseline replacement policy: pseudo LRU
Security Evaluation: RSA Attack

Baseline LRU: Access pattern of sqr, mul is clear

for \( i = n-1 \) down to 0 do
  \( r = \text{sqr}(r) \)
  ....
  if \( e_i == 1 \) then
    \( r = \text{mul}(r,b) \)
  end
end
Security Evaluation: RSA Attack

Baseline LRU: Access pattern of sqr, mul is clear

SHARP: No obvious access pattern of sqr, mul
L3 Misses Per Kilo Instructions

Inability to evict shared data causes cache thrashing

“cvb” performs the worst
Execution Time

- Modest slowdown of 6% due to large working set
- Average execution time increase ≈ 1%

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
More in the Paper

- Prevention of flush-based attacks
- Detailed evaluation
  - Mixes of SPEC workloads
  - Scalability to 8,16 cores
  - Threshold Alarm Analysis
- Handling of related attacks, defenses
- Insights into the scheme, corner cases
Conclusion

• Insight: Conflict-based attacks rely on “Inclusion Victims”

• Presented SHARP:
  – Shared cache replacement policy that defends against conflict-based attacks by preventing inclusion victims
  – Slightly modified “clflush” instruction to prevent flush-based attacks

✓ Prevents all known cache-based side channel attacks
✓ Minimal performance loss
✓ No programmer intervention
✓ Minor hardware modifications
Secure Hierarchy-Aware Cache Replacement Policy (SHARP): Defending Against Cache-Based Side Channel Attacks

Mengjia Yan, Bhargava Gopireddy, Thomas Shull, Josep Torrellas
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

ISCA 2017
Backup
- Starvation
- Threshold for alarms
- Pathological cases
- How does it apply to other replacement policies?
  - Performance will still be better?
Performance Evaluation on SPEC Benchmarks

Normalized L3 MPKI

Normalized IPC

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Performance Evaluation on Scalability

- Mixes of SPEC applications on 8-core setup
Performance Evaluation on Scalability

- PARSEC applications on 16-core setup
Alarm Analysis

- An attacker thread will increment its counter at least 100,000 times in 1 billion cycles
- It is safe to use 2,000 as threshold in SHARP4

### Alarm Analysis Table

<table>
<thead>
<tr>
<th>Appls.</th>
<th>cvb</th>
<th>query</th>
<th>SHARP1</th>
<th>SHARP2</th>
<th>SHARP3</th>
<th>SHARP4</th>
</tr>
</thead>
<tbody>
<tr>
<td>pov-mcf</td>
<td>285238</td>
<td>89</td>
<td>7285</td>
<td>171</td>
<td>121</td>
<td>84</td>
</tr>
<tr>
<td>lib-sje</td>
<td>1618715</td>
<td>1460</td>
<td>4747</td>
<td>2033</td>
<td>1820</td>
<td>1403</td>
</tr>
<tr>
<td>gob-mcf</td>
<td>549976</td>
<td>687</td>
<td>10001</td>
<td>1160</td>
<td>1426</td>
<td>1045</td>
</tr>
<tr>
<td>ast-pov</td>
<td>22701</td>
<td>19</td>
<td>1774</td>
<td>137</td>
<td>36</td>
<td>7</td>
</tr>
<tr>
<td>h26-gob</td>
<td>511</td>
<td>0</td>
<td>16</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bzi-sje</td>
<td>38669</td>
<td>7</td>
<td>177</td>
<td>9</td>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>h26-per</td>
<td>60536</td>
<td>1</td>
<td>974</td>
<td>184</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>cal-gob</td>
<td>132169</td>
<td>0</td>
<td>37</td>
<td>25</td>
<td>33</td>
<td>1</td>
</tr>
<tr>
<td>dea-pov</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>blackscholes</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bodytrack</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>canneal</td>
<td>153165</td>
<td>37</td>
<td>1192</td>
<td>39</td>
<td>43</td>
<td>37</td>
</tr>
<tr>
<td>dedup</td>
<td>145079</td>
<td>13</td>
<td>410</td>
<td>32</td>
<td>18</td>
<td>36</td>
</tr>
<tr>
<td>facesim</td>
<td>46409</td>
<td>12</td>
<td>97</td>
<td>32</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>ferret</td>
<td>91443</td>
<td>6</td>
<td>2097</td>
<td>102</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>25643</td>
<td>2</td>
<td>556</td>
<td>144</td>
<td>26</td>
<td>3</td>
</tr>
<tr>
<td>freqmine</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>raytrace</td>
<td>10013</td>
<td>1</td>
<td>85</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>swaptions</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x264</td>
<td>35897</td>
<td>2</td>
<td>423</td>
<td>10</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>

### Alarms per 1 billion cycles in benign workloads

| MAX     | 1618715 | 1460 | 10001 | 2033 | 1820 | 1403 |
Compare to Related Works

• Conflict-based attack
  – Cache partition
  – Software assisted cache locking

High performance overhead

• Flush-based attack
  – Disable “clflush” in user space

- Require code modification
- Difficult to precisely determine addresses to protect

Legacy issues
Performance Evaluation on PARSEC Benchmark

Inability to evict shared data causes cache thrashing, thus higher MKPI

Reducing inclusion victims lowers MPKI

Average MPKI increase is low

"cvb" performs the worst
Performance Evaluation on PARSEC Benchmark

Modest slowdown of 6% due to large working set.

Average execution time increase ≈ 1%
Experiment Setup

- MarssX86 cycle-level full system simulator

### Parameters for the simulated system

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multicore</td>
<td>2-16 cores at 2.5GHz</td>
</tr>
<tr>
<td>Core</td>
<td>4-issue, out-of-order, 128-entry ROB</td>
</tr>
<tr>
<td>Private L1 I-Cache/D-Cache</td>
<td>32KB, 64B line, 4-way</td>
</tr>
<tr>
<td></td>
<td>Access latency: 1 cycle</td>
</tr>
<tr>
<td>Private L2 Cache</td>
<td>256KB, 64B line, 8-way</td>
</tr>
<tr>
<td></td>
<td>Access latency: 5 cycles after L1</td>
</tr>
<tr>
<td>Query from L3 to L2</td>
<td>3 cycles network latency each way</td>
</tr>
<tr>
<td>Shared L3 Cache</td>
<td>2MB bank per core, 64B line, 16-way</td>
</tr>
<tr>
<td></td>
<td>Access latency: 10 cycles after L2</td>
</tr>
<tr>
<td>Coherence Protocol</td>
<td>MESI</td>
</tr>
<tr>
<td>DRAM</td>
<td>Access latency: 50ms after L3</td>
</tr>
<tr>
<td>Operating System</td>
<td>64-bit version of Ubuntu 10.4</td>
</tr>
</tbody>
</table>

### Evaluated configurations

<table>
<thead>
<tr>
<th>Config.</th>
<th>Line Replacement Policy in L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>Pseudo-LRU replacement.</td>
</tr>
<tr>
<td>cvb</td>
<td>Use CVBs in both step 1 and 2</td>
</tr>
<tr>
<td>query</td>
<td>CVBs in step1 &amp; queries in step 2</td>
</tr>
<tr>
<td>SHARPX</td>
<td>CVBs in step 1. In step 2, limit the max number of queries to X, where X = 1, 2, 3 or 4.</td>
</tr>
</tbody>
</table>
SHARP: New Cache Replacement for Security

- Prevents an attacker thread from creating “Inclusion Victims”

![Diagram showing cache hierarchy and attack prevention](image)

- Prevents all known cache-based side channel attacks
- Minimal performance loss
- No programmer intervention

Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Attacks on Inclusive Hierarchical Caches

- Cache based side channel attacks rely on “Inclusion Victims”

![Diagram showing cache eviction and inclusion victim concept]
SHARP: New Cache Replacement for Security

- Prevents an attacker thread from creating “Inclusion Victims”

Prevents cache-based side channel attacks
minimal performance penalty
Sample Attack – RSA Encryption Key

Square and multiply based exponentiation

- **Input**: base $b$, modulo $m$, exponent $e = (e_{n-1} ... e_0)_2$
- **Output**: $b^e \mod m$

$r = 1$

**for** $i = n-1$ **down to** 0 **do**

- $r = \text{sqr}(r)$
- $r = \text{mod}(r, m)$
- **if** $e_i == 1$ **then**
  - $r = \text{mul}(r, b)$
  - $r = \text{mod}(r, m)$
- **end**

**end**

**return** $r$

Probe addresses used by spy