Detection of Cache Side Channel Attacks using Thread Level Monitoring of Hardware Performance Counters

Pavitra Prakash Bhade
pavitra19231101@iitgoa.ac.in

Sharad Sinha
sharad@iitgoa.ac.in

School of Mathematics and Computer Science
Indian Institute of Technology Goa (IIT Goa), India
Outline

• Problem Statement
• Cache Memory: Introduction and Architecture
• Cache Attack: Flush + Reload Attack
• Proposed Technique
• Experiments and Results
• Conclusion
Problem Statement

• Modern microprocessors have cache memory structure that is vulnerable to cache side channel attacks

• Directly moving to mitigation of such attacks, have the following issues:
  • Need hardware modification in the architecture
  • Software upgrades have certain performance impact
  • Type of mitigation may be dependent on the type of attack

• Hence, “Detection” ensures that mitigation is applied on need basis, to serve as first line of defence against such attacks
Cache Memory: Structure

- Fast and small memory for frequent accesses by the processor.
- Most multicore architectures (ex. Intel x86) have hierarchical cache levels.
- Level 1 and Level 2 are individual per core, whereas the last level cache (LLC) is common across all cores.
- Last Level Cache is inclusive in Intel architecture:
  - Element evicted from LLC is evicted from other levels as well.

Fig: Intel Sandy Bridge Cache Architecture

Pavitra Prakash Bhide, Sharad Sinha (IIT Goa)
IEEE 14th Intl. Symp. on MCSoC, 2021, Singapore
Cache Memory : Timing analysis

- Presence and absence of data / instructions in the cache causes cache hit or cache miss respectively.
- The time difference between cache hit and miss, can be analysed by an attacker to trace the victim activity and leak secrets from the shared cache.

Fig: Difference in Cache Hit and Cache Miss Timings
Flush + Reload Attack

- Consider this square and multiply algorithm performing encryption/decryption function.
- Steps 8 and 9 are dependent on the value of e.
- If e is 1, 8 and 9 will be executed.
- Else directly 10 will be executed.
- Monitoring the cache hit/miss for instructions 8 and 9, reveals the value of e.
- Cache hit: e=1
- Cache miss: e=0

```
1 function exponent(b, e, m)
2 begin
3 x ← 1
4 for i ← |e| − 1 downto 0 do
5 x ← x²
6 x ← x mod m
7 if (e_i = 1) then
8 x ← xb
9 x ← x mod m
10 endif
11 done
12 return x
13 end
```

Fig: Square and multiply algorithm for encryption/decryption.
Detection of Cache Attacks

- Cache Side Channel attacks do not cause any architectural changes.
- But these attacks leave certain microarchitectural traces, which can be monitored for the purpose of detection.
- Hardware performance counters can be monitored to detect suspicious activities if the counts go high.
Hardware Performance Counters (HPC)

• Eviction based cache attacks like Flush + Reload cause the data or instructions to forcefully be evicted from the cache

• This causes certain hardware performance counters like Cache hits, Cache misses, etc to show exorbitantly high counts

• Such counters can be monitored to detect the attack
Problem with system level HPCs

• The applications running on the system can sometimes mimic attack like scenario.

• Multiple genuine applications involving highly random memory accesses can cause high cache miss and other counts, which may give rise to false alarms in detection

• Such false alarms lead to unnecessary application of mitigation techniques, further impacting the performance
Proposed Technique: Thread Level HPC

- The Hardware performance counters can be fine grained to thread level, instead of the system level.
- Some section of the victim code is confidential and holds secrets.
- Protecting only this section is important, rather than the entire application or the system.
- This reduces monitoring overhead as well as chances of false alarms.
- Hence only the thread performing the execution of this sensitive section should be monitored.
Implementation Details

• Operating System
  • Ubuntu 20.04 LTS

• Attack Performed
  • Cache Template Attack
    • Based on Flush + Reload method
    • Extracts keypresses of the victim
  • Three editors
    • Sublime Text
    • Gedit
    • Terminal

• Monitoring tool
  • Vtune Profiler
    • Filter using thread id
Proposed Framework

- Victim Process
- Vtune Profiler
- Monitoring Process
- Attack Process

Thread To be Monitored
Underlying Micro-Architecture
Experiments and Results

• Four Event Counters have shown exorbitantly high counts on an event of attack
  • DTLB LOAD MISS
  • MEM LOAD RETIRED L1 MISS
  • MEM LOAD RETIRED L3 HIT
  • DTLB STORE MISSES

• The experiments were conducted in three scenarios:
  • Attack (When there is a victim and attacker code running)
  • No Attack (When there is only victim code running)
  • No Attack system overloaded (When the victim code is running with other heavy applications running in the background, no attacker code)
Experiments and Results: DTLB LOAD MISS

System level

Thread Level

Pavitra Prakash Bhade, Sharad Sinha (IIT Goa)
IEEE 14th Intl. Symp. on MCMoC, 2021, Singapore
Experiments and Results : MEM LOAD
RETIRED L1 MISS

System level

Thread Level

Pavitra Prakash Bhade, Sharad Sinha (IIT Goa)
IEEE 14th Intl. Symp. on MCSoc, 2021, Singapore
Experiments and Results: MEM LOAD RETIRED L3 HIT

System level

Thread Level

Pavitra Prakash Bhade, Sharad Sinha (IIT Goa)
IEEE 14th Intl. Symp. on MCSoc, 2021, Singapore
Experiments and Results : DTLB STORE MISS

System level

Thread Level
## False positives at System v/s Thread level

<table>
<thead>
<tr>
<th>EVENT COUNT</th>
<th>GEDIT (SYSTEM LEVEL)</th>
<th>TERMINAL (SYSTEM LEVEL)</th>
<th>SUBLIME TEXT (SYSTEM LEVEL)</th>
<th>ALL THREE EDITORS (THREAD LEVEL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTLB LOAD MISS</td>
<td>60</td>
<td>52</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>MEMORY LOAD RETIRED L1 MISS</td>
<td>44</td>
<td>37</td>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>MEMORY LOAD RETIRED L3 HIT</td>
<td>40</td>
<td>38</td>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>DTLB STORE MISS</td>
<td>48</td>
<td>43</td>
<td>48</td>
<td>0</td>
</tr>
</tbody>
</table>

Thread level monitoring shows nil false positives, whereas system level shows 48% false positives
False negatives at System v/s Thread level

- The highest count value in case of High CPU Intensive application scenario, acts as the lower limit for the threshold value.

- Anything above that indicates attack.

- This derivation becomes very evident in case of thread level monitoring, giving rise to zero false negatives, as against non zero false negatives in system level monitoring.
Conclusion

- Thread level monitoring:
  - Reduces false alarms
  - Reduces performance impact due to unnecessary application of mitigation techniques
  - Reduces monitoring overhead, as only sensitive thread is being monitored and not the entire system.
  - Results of thread level monitoring are more clear to derive threshold, further reducing false negatives
References


