Cache Side-Channel Attacks
(Brief Introduction)

Presenter: Aria Shahverdi

9/31/2019
How do we load data from Main Memory?
Memory Locality

- Future memory accesses are near past memory accesses
- Memories take advantages of two locality
  - Temporal Locality: near in time
    - We will often access the same data again very soon
  - Spatial Locality: near in space/distance
    - Our next access is often very close to our last access (or recent accesses)

```c
for(i = 0; i < 20; i++)
a[i] = a[i] * 2;
```
Cache Architecture High Level

Main Memory

Cache

Set 1
Set 2
Set 64

CPU

Main Memory

Address

Data
Set Associative Cache Architecture

8-way set associative Cache

Set 1
- line1
- line2
- line3
- line4
- line5
- line6
- line7
- line8

Set 2
- line1
- line2
- line3
- line4
- line5
- line6
- line7
- line8

Set 3
- line1
- line2
- line3
- line4
- line5
- line6
- line7
- line8

Set 64
- line1
- line2
- line3
- line4
- line5
- line6
- line7
- line8

Main Memory
Introduction to Cache Architecture
Cache Architecture (Summary)

• Unit of Memory in cache is a line
• A cache consists of multiple sets which stores fixed number of lines
• The number of lines in a set is called associativity
  • L1 is 8-way, L2 is 4-way, L3 is 12-way
• Last Level Cache (LLC) is inclusive
  • LLC contains copies of all of the data in the lower cache level
  • Evicting data from LLC remove that data from all other cache levels
Accessing Memory (Cache Hit)

Fast Access to Data
Accessing Memory (Cache Miss)

Slow Access to Data
Cache Hit vs. Miss Time Difference

• ≈10 Million measurement
Cache Attack Model

: Attacker

: Victim
Some Cache Attack Technique

• Evict and Time
• Flush and Reload
• Prime and Probe
Some Cache Attack Technique

• Evict and Time
• Flush and Reload
• Prime and Probe
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
How do we fill a cache set?

• By Accessing some of the memory locations the corresponding locations in the cache is going to be filled.

• Main Challenge: which lines to access?
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
1. Trigger encryption

2. Selectively manipulate the state of the cache (e.g. evict a full cache set)

3. Trigger encryption

4. Measure how long it took

5. Deduce what cache sets it accessed

6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Evict and Time

1. Trigger encryption
2. Selectively manipulate the state of the cache (e.g. evict a full cache set)
3. Trigger encryption
4. Measure how long it took
5. Deduce what cache sets it accessed
6. Repeat step 1-4 to gain information on all the set the encryption accessed
Some Cache Attack Technique

• Evict and Time
• Flush and Reload
• Prime and Probe
Flush and Reload

• Exploits cache behavior to leak information on victim access to shared memory.
  • Shared libraries
  • Memory de-duplication

• Spy monitors victim’s access to shared code
  • Spy can determine what victim does
  • Spy can infer the data the victim operates on
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat
Flush a Line From Cache
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat
Reload a Line From Cache

Main Memory

No Access by Victim

reload
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat

Slow means no access by victim
Reload a Line From Cache
Flush and Reload

1. Flush memory line
2. Wait a bit
3. Measure time to Reload line
4. Repeat

Fast means that victim accessed
Some Cache Attack Technique

• Evict and Time
• Flush and Reload
• Prime and Probe
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory

2. Victim executes and evicts some of the cache lines

3. Attacker accesses those cache line and measure time
Fill a cache set (In this example 2 sets)
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time
Victim Execution

Main Memory

- Blue: Attacker’s data
- Red: Victim’s data

Access some lines
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time
Probe

: Attacker’s data

: Victim’s data

Probe a set
Prime and Probe

1. Attacker fills a set with its own data by accessing some locations in memory
2. Victim executes and evicts some of the cache lines
3. Attacker accesses those cache line and measure time

Fast Access: Not accessed by the victim

Slow Access: Accessed by the victim
How To Recover Secret Key?

• How do we compute $b^e \mod n$?
  • Assume $e$ is secret information we want to recover.

• Bit = 0 : Square

• Bit = 1 : Square + Multiply

✓ The Sequence of operation will reveal the secret information.

```
x ← 1
for i ← |e| - 1 downto 0 do
    x ← x^2 \mod n
    if (e_i = 1) then
        x = xb \mod n
    endif
endfor
done
return x
```
A Sample Measurement (Flush and Reload)
Cache Attack on Database

- User execute queries in the form of range queries
  - Asks for entries with column value between value1 and value2
  - Attacker sees the volume of the responses

```
SELECT *
FROM table
WHERE column BETWEEN value1 AND value2;
```
Cache Attack on Database

SELECT * FROM table WHERE Grade BETWEEN 1 AND 1

SELECT * FROM table WHERE Grade BETWEEN 1 AND 2

SELECT * FROM table WHERE Grade BETWEEN 3 AND 4

<table>
<thead>
<tr>
<th>Students</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
</tbody>
</table>
Cache Attack on Database

SELECT * FROM `table` WHERE Grade BETWEEN 1 AND 1

SELECT * FROM `table` WHERE Grade BETWEEN 1 AND 2

SELECT * FROM `table` WHERE Grade BETWEEN 3 AND 4

<table>
<thead>
<tr>
<th>Students</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
</tbody>
</table>
Cache Attack on Database

```
SELECT * FROM table WHERE Grade BETWEEN 1 AND 1

SELECT * FROM table WHERE Grade BETWEEN 1 AND 2

SELECT * FROM table WHERE Grade BETWEEN 3 AND 4
```

<table>
<thead>
<tr>
<th>Students</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>3</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
</tr>
<tr>
<td>G</td>
<td>3</td>
</tr>
<tr>
<td>H</td>
<td>4</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
</tbody>
</table>
Attacker View From Cache

- Counts how many times the server executes the line corresponding to returning an entry.
- From that, the attacker figures out approximately the value of each volume.
Noisy Volume Recovered

Peaks represent the Volumes
How to reconstruct the database?

• \([1-2] = [1-1] + [2-2]\)
• \([1-3] = [1-2] + [3-3] = [1-1] + [2-3]\)
• \([1-4] = [1-1] + [2-4] = \ldots\)
[1-1] = 2
[2-2] = [1-2] - [1-1] = 2
[3-3] = [1-3] - [1-2] = 15
[4-4] = [1-4] - [1-3] = 5
What about the case for measurements from cache

• It is harder because the measurement from cache are noisy
• Some of the volumes might be missing
  • Some of the connections ins graph is missing
• There might be some extra volumes in the graph
  • There are extra nodes in the graph which should not be there
• We still can recover the database in some of the cases
Side Channel Attacks Examples

• Timing Attacks
  • Cache Attack
• Power Analysis Attack
• Electromagnetic Emissions
• Acoustic Emission
• Fault Attacks
Thank You