Cache Side Channel Attack: Exploitability and Countermeasures

Gorka Irazoqui
Xiaofei (Rex) Guo, Ph.D.
girazoki *noSPAM* wpi.edu
xiaofei.rex.guo *noSPAM* tetrationanalytics.com
Who are We?

• Gorka Irazoqui
  • PhD candidate in WPI
  • Intern at Intel in summer 2016
  • Focus on micro-architectural attacks
Who are We?

• Xiaofei (Rex) Guo
  • Technical lead at Cisco Tetration Analytics
    • Visibility to everything in data center in real time
    • Automated and dynamic policy generation and enforcement
  • Worked at Intel Security Center of Excellence and Qualcomm Product Security Initiative
    • IoT and mobile platform security, infrastructure security, and application security
  • PhD from New York University
We don’t speak for our employer. All the opinions and information here are our responsibility including mistakes and bad jokes.
“You must be kidding, cache attacks are not practical!”
“You must be kidding, cache attacks are not practical!”
“You must be kidding, cache attacks are not practical!”

Security considerations and disallowing inter-Virtual Machine Transparent Page Sharing (2080735)

Purpose

This article acknowledges the recent academic research that leverages Transparent Page Sharing (TPS) to gain unauthorized access to data under certain highly controlled conditions and documents VMware's precautionary measure of restricting TPS to individual virtual machines by default in upcoming ESX releases. At this time, VMware believes that the published information disclosure due to TPS between virtual machines is impractical in a real world deployment.
“You must be kidding, cache attacks are not practical!”
“You must be kidding, cache attacks are not practical!”

Security considerations and disallowing inter-Virtual Machine Transparent Page Sharing (2080735)

Purpose

This article acknowledges the recent academic research that leverages Transparent Page Sharing (TPS) to gain unauthorized access to data under certain highly controlled conditions and documents VMware’s precautionary measure of restricting TPS to individual virtual machines by default in upcoming ESXi releases. At this time, VMware believes that the published information disclosure due to TPS between virtual machines is impractical in a real world deployment.

CacheBleed OpenSSL Vulnerability Affects Intel-Based Cloud Servers

Only Sandy Bridge (and earlier) Intel CPUs are affected

Mar 2, 2016 08:26 GMT - By Catalin Cimpanu

Yesterday’s OpenSSL updates (1.0.2g and 1.0.1s) not only brought a fix against the already infamous DROWN attack but also patched seven other security flaws, one labeled as high, one moderate, and five as low severity.
Cache Attacks and Countermeasures: The Case of AES

Intel, Spark, AMD | Linux | OpenSSL AES
Feasibility Trend

Cache Attacks and Countermeasures: The Case of AES

Yet another MicroArchitectural Attack: exploiting I-Cache.

Intel, Spark, AMD | Linux | OpenSSL AES

Intel | Linux | OpenSSL RSA
Feasibility Trend

Cache Attacks and Countermeasures: The Case of AES

Yet another MicroArchitectural Attack: exploiting I-Cache.

FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack

- Intel, Spark, AMD | Linux | OpenSSL AES
- Intel | Linux | OpenSSL RSA
- Intel (Cross-core) | Linux (deduplication) | GnuPG RSA
Cache Attacks and Countermeasures: The Case of AES

Yet another MicroArchitectural Attack: exploiting I-Cache.

FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack

Last-Level Cache Side-Channel Attacks are Practical

Intel, Spark, AMD | Linux | OpenSSL AES

Intel | Linux | OpenSSL RSA

Intel (Cross-core) | Linux (deduplication) | GnuPG RSA

Intel (Cross-Core) | Linux (no deduplication) | GnuPG RSA
Cache Attacks and Countermeasures: The Case of AES

Yet another MicroArchitectural Attack: exploiting I-Cache.

FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack

Last-Level Cache Side-Channel Attacks are Practical

Cross Processor Cache Attacks
Cache Attacks and Countermeasures: The Case of AES

Yet another MicroArchitectural Attack: exploiting I-Cache.

FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack

Last-Level Cache Side-Channel Attacks are Practical

Cross Processor Cache Attacks

ARMageddon: Last-Level Cache Attacks on Mobile Devices

Intel, Spark, AMD | Linux | OpenSSL AES

Intel | Linux | OpenSSL RSA

Intel (Cross-core) | Linux (deduplication) | GnuPG RSA

Intel (Cross-Core) | Linux (no deduplication) | GnuPG RSA

AMD (cross CPU) | Linux | OpenSSL AES and GnuPG El Gamal

ARM (cross core/CPU) | Android | Bouncy Castle AES
Functionality
LLC as a Side Channel?

- Caches: fast access memories
- Why would an attacker use LLC as covert channel?
  - Cross-core
  - Inclusiveness
  - High resolution
Cache Architecture

- Set associative: cache divided in n-way sets
- Location in the cache determined by physical address
Flush + Reload Attack

- Requirement 1: deduplication
  - Identical read-only memory pages are shared
  - Attacker and victim access the same address
  - Linux and KVM (KSM), Vmware (TPS) and Android (Zygote)

- Requirement 2: flush instruction (e.g., clflush in x86)

- CVE 2014-3356: Vmware enabled deduplication by default
Flush + Reload Attack

- Attacker flushes a cached memory location
Flush + Reload Attack

- Attacker flushes a cached memory location
Flush + Reload Attack

• Attacker flushes a cached memory location
• Victim accesses/does not access
Flush + Reload Attack

- Attacker flushes a cached memory location
- Victim accesses/does not access
- Attacker re-accesses memory location
  - Fast access time -> victim accessed
  - Slow access time -> victim did not access
Flush + Reload Attack Summary

• **Pros:**
  • Low noise: focus on one line, noisy process needs to fill an entire set
  • Applicable across CPU sockets! Flush instruction invalidates memory in other CPUs
  • Works in non-inclusive caches

• **Cons:**
  • Requirement might be met in some scenarios
  • Can only recover statically allocated data
Evict + Reload Attack

- No flush instruction?
- Attacker needs to evict data from LLC
- Attacker can use huge pages
- Physical address selects the set to occupy
Evict + Reload Attack

- No flush instruction?
- Attacker needs to evict data from LLC
- Attacker can use huge pages
- Physical address selects the set to occupy
- Attacker evicts (fills set)
Evict + Reload Attack

- No flush instruction?
- Attacker needs to evict data from LLC
- Attacker can use huge pages
- Physical address selects the set to occupy
- Attacker evicts (fills set)
Evict + Reload Attack

- No flush instruction?
- Attacker needs to evict data from LLC
- Attacker can use huge pages
- Physical address selects the set to occupy
- Attacker evicts (fills set)
- Victim accesses/does not access
Evict + Reload Attack

- No flush instruction?
- Attacker needs to evict data from LLC
- Attacker can use huge pages
- Physical address selects the set to occupy
- Attacker evicts (fills set)
- Victim accesses/does not access
- Attacker reloads
  - Fast access time -> victim accessed
  - Slow access time -> victim did not access
Evict + Reload Attack Summary

• Pros:
  • Applicable in processors without flush instruction (e.g. most ARM processors)

• Cons:
  • Can only target statically allocated memory
  • Deal with LLC slices (undocumented)
  • Only works with inclusive caches
  • Only works in the same CPU socket
Prime + Probe Attack

- No shared memory pages?
  - Attacker can know the set utilized by the victim
- Attacker Primes
Prime + Probe Attack

- No shared memory pages?
  - Attacker can know the set utilized by the victim
- Attacker Primes
- Victim access/es/ accesses
Prime + Probe Attack

- No shared memory pages?
  - Attacker can know the set utilized by the victim
- Attacker Primes
- Victim accesses/not accesses
- Attacker re-accesses
  - Fast access time -> victim accessed
  - Slow access time -> victim did not access
Prime + Probe Attack Summary

• Pros
  • Does not need shared memory! (Broader impact)
  • Can target static and dynamically allocated memory!

• Cons:
  • Noisier than Flush + Reload
  • Dealing with LLC slices (undocumented)
  • Only works with inclusive caches
  • Only works in the same CPU socket
  • Need to identify the target set
How to retrieve information?

Montgomery ladder RSA

```
1 function modpow (a, b);
   Input : base b, modulus N, secret
            E = (e_{k-1}, ..., e_1, e_0)
   Output: b^E \mod N
2     R_0 = 1; R_1 = b;
3     for i = k - 1 downto 0 do
4        if e_i == 0 then
5            R_1 = R_0 * R_1 \mod N;
6            R_0 = R_0 * R_0 \mod N;
7        end
8        else
9            R_0 = R_0 * R_1 \mod N;
10           R_1 = R_1 * R_1 \mod N;
11       end
12 end
13 return R_0;
```

Flush and Reload Cache

Physical address
P=0x7fffc480
How to retrieve information?

Montgomery ladder RSA

```plaintext
function modpow (a, b);
    Input: base b, modulus N, secret
    Output: b^E mod N
    R_0 = 1; R_1 = b;
    for i = k-1 downto 0 do
        if e_i == 0 then
            R_1 = R_0 * R_1 mod N;
            R_0 = R_0 * R_0 mod N;
        else
            R_0 = R_0 * R_1 mod N;
            R_1 = R_1 * R_1 mod N;
        end
    end
    return R_0;
```

Flush and Reload Cache

Physical address
P=0x7fff4c80
How to retrieve information?

Montgomery ladder RSA

```
function modpow (a, b);
  \textbf{Input} : base \( b \), modulus \( N \), secret \\
  \quad E = (e_{k-1}, \ldots, e_1, e_0) \\
  \textbf{Output} : b^E \mod N

1. \( R_0 = 1; R_1 = b; \)
2. \( k \) - 1 downto 0 do
3. \quad if \( e_i = 0 \) then
4. \quad \quad \quad R_1 = R_0 \times R_1 \mod N; \\
5. \quad \quad \quad R_0 = R_0 \times R_0 \mod N; \\
6. \quad else
7. \quad \quad \quad R_0 = R_0 \times R_1 \mod N; \\
8. \quad \quad \quad R_1 = R_1 \times R_1 \mod N; \\
9. \quad end
10. end
11. return \( R_0 \);
```

Flush and Reload Cache

Physical address

P=0x7fff4c80
How to retrieve information?

Montgomery ladder RSA

```
function modpow (a, b);
Input : base b, modulus N, secret
E = (e_{k-1}, ..., e_1, e_0)
Output: b^E mod N
R_0 = 1; R_1 = b;
for i = k - 1 downto 0 do
  if e_i == 0 then
    R_1 = R_0 * R_1 mod N;
    R_0 = R_0 * R_0 mod N;
  else
    R_0 = R_0 * R_1 mod N;
    R_1 = R_1 * R_1 mod N;
end
end
return R_0;
```

Flush and Reload Cache

Physical address
P=0x7fff480
How to retrieve information?

Montgomery ladder RSA

```plaintext
function modpow (a, b);
    Input: base b, modulus N, secret e
    Output: b^e mod N
    R_0 = 1; R_1 = b;
    for i = k - 1 downto 0 do
        if e_i == 0 then
            R_1 = R_0 * R_1 mod N;
            R_0 = R_0 * R_0 mod N;
        else
            R_0 = R_0 * R_1 mod N;
            R_1 = R_1 * R_1 mod N;
        end
    end
    return R_0;
```

Flush and Reload Cache

Physical address
P=0x7fffec480
How to retrieve information?

Montgomery ladder RSA

```
function modpow (b, a, N);
    \textbf{Input}: base \( b \), modulus \( N \), secret \( E = (e_{k-1}, \ldots, e_1, e_0) \)
    \textbf{Output}: \( b^E \mod N \)
    \textbf{R_0} = 1; \textbf{R}_1 = b;
    \textbf{for } i = k - 1 \textbf{ downto } 0 \textbf{ do}
        \textbf{if } e_i = 0 \textbf{ then}
            R_1 = R_0 \times R_1 \mod N;
            R_0 = R_0 \times R_0 \mod N;
        \textbf{end}
        \textbf{else}
            R_0 = R_0 \times R_1 \mod N;
            R_1 = R_1 \times R_1 \mod N;
        \textbf{end}
    \textbf{end}
    \textbf{return } R_0;
```

Prime and Probe Cache

Physical address
\( P = 0x7ffffff480 \)
How to retrieve information?

Montgomery ladder RSA

```python
1 function modpow (a, b):
    Input : base b, modulus N, secret
    Output: b^E mod N
    E = (e_{k-1}, ..., e_1, e_0)
    R_0 = 1; R_1 = b;
    for i = k - 1 downto 0 do
        if e_i == 0 then
            R_1 = R_0 * R_1 mod N;
        else
            R_0 = R_0 * R_0 mod N;
            R_1 = R_1 * R_1 mod N;
        end
    end
    return R_0;
```

Prime and Probe Cache

Physical address
P=0x7fff480
How to retrieve information?

Montgomery ladder RSA

```
function modpow (a, b);
  Input : base b, modulus N, secret
          \[ E = (e_{k-1}, \ldots, e_1, e_0) \]
  Output: \( b^E \mod N \)
  \[ R_0 = 1; R_1 = b; \]
  for \( i = k - 1 \) downto 0 do
    if \( e_i = 0 \) then
      \[ R_1 = R_0 \times R_1 \mod N; \]
      \[ R_0 = R_0 \times R_0 \mod N; \]
    else
      \[ R_0 = R_0 \times R_1 \mod N; \]
      \[ R_1 = R_1 \times R_1 \mod N; \]
  end
end
return R_0;
```
## Attack Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Flush + Reload</th>
<th>Evict + Reload</th>
<th>Prime + Probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require Memory Deduplication</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Require flush instruction</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Attack memory type</td>
<td>static</td>
<td>static</td>
<td>static + dynamic</td>
</tr>
<tr>
<td>Noise</td>
<td>low</td>
<td>low</td>
<td>high</td>
</tr>
</tbody>
</table>
Applicability
VMs share underlying hardware

Hardware isolation is usually not provided

Example RSA in Amazon EC2 [INCI16]

Pros:

- Own virtualized OS. Access to timers or huge pages
- If deduplication enabled, both attacks are applicable

Cons:

- Requires co-residency of VMs
- High amount of noise
IaaS/PaaS Cloud Infrastructures

- How to find co-residency?
- Use available information!
- Profile the target LLC accesses
- Does the cache trace match the trace we expect?
  - If yes, co-residency
  - If no, open more VMs
- Other mechanisms utilize memory bus locking attacks
- Example RSA exponentiations easily distinguishable
Demo: AES Key Recovery Across VMs

- We utilize KVM hypervisor
- Server using T-table AES (T-tables shared)
- Server encrypting plaintext with unknown key
- Attacker requests decryptions and recovers the key
- We check whether the entries of the T-tables have been used
- We XOR with the ciphertext after doing statistics to get the key

\[ T_0 = \{0x63, 0x7b, 0x7c, 0x6f\} \]
\[ C_0 = \{0x63, 0x7b, 0x7c, 0x6f\} \]
Browser Javascript

- Attacker embeds JS into the website
- Victim accesses the website
- Victim’s browser executes the JS
- Example: Incognito browsing profiling [OREN15]

Pros:
- No need to find co-resident target
- Attack executed in local machine (although sandboxed)

Cons:
- Flush and Reload can not be applied
- Fine grain timers hard to achieve
Smart Phone Applications

- Smartphone applications are logically isolated by the OS
- However, as with TEEs, all applications utilize the hardware caches
- Micro-architectural attacks look as innocent binaries, as they only perform timed memory accesses
- Example: AES key steal across apps [LIPP16]
Smart Phone Applications

• **Pros:**
  • Deduplication is generally used (e.g. Android)
  • Easy deployment

• **Cons**
  • Flush instruction has to be enabled by SoC (only Samsung S6 for now)
  • Pseudo Random Replacement policies (reverse engineered)
  • Device dependent algorithms (e.g. non-inclusive caches or lockdown)
Trusted Execution Environment

- Trusted execution environments designed to achieve isolation from untrusted processes
- But both trusted and untrusted environments access same hardware caches!
- Enclave to enclave or host to enclave attacks are possible
- Example: TrustZone AES key steal [BRM15]
- Example: Intel SGX RSA key steal [SCW17]

![Diagram showing the TEE Enclave and Untrusted process with their respective caches and DRAM compartments.](image-url)
Trust Execution Environment

• **Pros**
  • Higher resolution: The OS can be malicious! more fine grain resources (including scheduling)
  • No need to find co-resident target
  • Limited noise: malicious OS can interrupt processes after (virtually) every memory access

• **Cons**
  • Flush and Reload not applicable (deduplication disabled)
Trust Execution Environment

• Pros
  • Higher resolution: The OS can be malicious! more fine grain resources (including scheduling)
  • No need to find co-resident target
  • Limited noise: malicious OS can interrupt processes after (virtually) every memory access

• Cons
  • Flush and Reload not applicable (deduplication disabled)
Trust Execution Environment

• Pros
  • Higher resolution: The OS can be malicious! more fine grain resources (including scheduling)
  • No need to find co-resident target
  • Low noise: malicious OS can interrupt processes after (virtually) every memory access
• Cons
  • Flush and Reload not applicable (deduplication disabled)
Countermeasures
Design Cache Leakage Free Code

- Secret independent instruction accesses
- Secret independent data accesses
- Identification of variables that contain information related to the secret (manual inspection, taint analysis, etc.)
- Obtain cache timing traces to correlate with the secret variables to measure the leakage
Design Cache Leakage Free Code

```
function modpow (a, b);
    Input : base b, modulus N, secret
    E = (e_{k-1}, ..., e_1, e_0)
    Output: b^E \mod N

2    R_0 = 1; R_1 = b;
3    for i = k - 1 downto 0 do
4        if e_i==0 then
5            R_1 = R_0 \ast R_1 \mod N;
6            R_0 = R_0 \ast R_0 \mod N;
7        end
8        else
9            R_0 = R_0 \ast R_1 \mod N;
10           R_1 = R_1 \ast R_1 \mod N;
11        end
12    end
13    return R_0;
```
Design Cache Leakage Free Code

function modpow \((a, b)\);

\[
\text{Input: base } b, \text{ modulus } N, \text{ secret } E = (e_{k-1}, \ldots, e_1, e_0)
\]

\[
\text{Output: } b^E \mod N
\]

1. \(R_0 = 1; \ R_1 = b;\)
2. \(\text{for } i = k - 1 \text{ downto } 0 \text{ do}\)
3. \(\text{if } e_i = 0 \text{ then}\)
4. \(\quad R_1 = R_0 \times R_1 \mod N;\)
5. \(\quad R_0 = R_0 \times R_0 \mod N;\)
6. \(\text{end}\)
7. \(\text{else}\)
8. \(\quad R_0 = R_0 \times R_1 \mod N;\)
9. \(\quad R_1 = R_1 \times R_1 \mod N;\)
10. \(\text{end}\)
11. \(\text{end}\)
12. \(\text{return } R_0;\)

CVE-2016-7439

secret dependent instruction access
Design Cache Leakage Free Code

CVE-2016-7439

```
function modpow (a, b);
Input : base b, modulus N, secret 
E = (e_{k-1}, ..., e_1, e_0)
Output: \( b^E \mod N \)

1) \( R[0] = 1; \ R[1] = b; \)
2) for \( i = k - 1 \) downto 0 do
3) \( R[\hat{e}_i] = R[0] \ast R[1] \mod N; \)
4) \( R[e_i] = R[e_i] \ast R[e_i] \mod N; \)
5) end
6) return \( R[0]; \)
```
Design Cache Leakage Free Code

function modpow (a, b);

Input : base b, modulus N, secret

\[ E = (e_{k-1}, \ldots, e_1, e_0) \]

Output: \( b^E \mod N \)

1. \( R[0] = 1; R[1] = b; \)
2. for \( i = k - 1 \) downto 0 do
3. \( R[e_i] = R[0] \times R[1] \mod N; \)
4. \( R[e_i] = R[e_i] \times R[e_i] \mod N; \)
5. end
6. return \( R[0]; \)

CVE-2016-7439

Secret independent instruction access

Secret dependent data access
Design Cache Leakage Free Code

```plaintext
function modpow (a, b);
    Input : base b, modulus N, secret
              E = (e_{k-1}, ..., e_1, e_0)
    Output: b^E mod N
2     R[0] = 1; R[1] = b;
3     for i = k - 1 downto 0 do
4         R[0] * e_i + R[1] * \hat{e}_i = R[0] * R[1] mod N;
5         R[1] * e_i + R[0] * \hat{e}_i =
              R[1] * R[1] * e_i + R[0] * R[0] * \hat{e}_i mod N;
6     end
7     return R[0];
```

Secret independent
instruction access

Secret independent
data access
Page Coloring

- Avoiding collisions in the LLC
- Location in LLC determined by physical address
- Give each user a color (address bits)
Cache Allocation Technology

- Intel CAT provides hardware framework to lock the cache
- Allows OS/hypervisor to mark cache ways as un-evictable
- Attacker can not influence victim’s cache accesses
- Modify hypervisor to support more lock partitions [LIU16]
Cache Allocation Technology

- Intel CAT provides hardware framework to lock the cache
- Allows OS/hypervisor to mark cache ways as un-evictable
- Attacker can not influence victim’s cache accesses
- Modify hypervisor to support more lock partitions [LIU16]
Cache Allocation Technology

- Intel CAT provides hardware framework to lock the cache
- Allows OS/hypervisor to mark cache ways as un-evictable
- Attacker cannot influence victim’s cache accesses
- Modify hypervisor to support more lock partitions [LIU16]
Behavior Detection

- Hardware Performance Counters (HPCs) can track hardware events (e.g. LLC misses)
- LLC attacks leave a clear trace in terms of cache misses/hits
- Hypervisor/OS tracks this events to detect unusual behavior
- Detection can be improved by inspecting memory access
## Countermeasure Comparison (Requirements)

<table>
<thead>
<tr>
<th></th>
<th>Leakage Free Code</th>
<th>Page Coloring</th>
<th>Intel CAT</th>
<th>Behavior Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Require source code change</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Require OS (hypervisor) update</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Depends</td>
</tr>
<tr>
<td>Require new hardware</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
</tbody>
</table>
## Countermeasure Comparison (Coverage)

<table>
<thead>
<tr>
<th></th>
<th>Leakage Free Code</th>
<th>Page Coloring</th>
<th>Intel CAT</th>
<th>Behavior Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>IaaS/PaaS</td>
<td>Y</td>
<td>Y</td>
<td>Depends</td>
<td>Y</td>
</tr>
<tr>
<td>Javascript in browser</td>
<td>Y</td>
<td>Depends</td>
<td>Depends</td>
<td>Y</td>
</tr>
<tr>
<td>Smartphone</td>
<td>Y</td>
<td>Y</td>
<td>Depends</td>
<td>Y</td>
</tr>
<tr>
<td>TEE</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>
Key Takeaways

• Cache attacks are complex but a real threat!
• Flush+Reload, Evict+Reload, Prime+Probe
• IaaS/PaaS, web browsers, smartphones, TEE,...What else?
• Call to action:
  • Application level: introduce cache leakage free code design
  • Hypervisor/OS level: page coloring for cache isolation
  • System level: use software to leverage hardware features (Intel CAT, performance counters)
References


