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
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)
Sample Cache Side-Channel Attack: RSA Encryption Key

```
for i = n-1 down to 0 do
    r = sqr(r)
    ....
    if e_i == 1 then
        r = mul(r, b)
    ....
end
```

Probe addresses: Monitored by the spy to obtain information
Attack Illustration

Victim L1 Cache

Spy L1 Cache

Cache Set

Shared L2 Cache (Inclusive)
Attack Illustration

Victim L1 Cache  Spy L1 Cache

Shared L2 Cache (Inclusive)

Probe address
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Attack Illustration

Evict

Conflict

Evict

Inclusion Victim

Spy's line

Probe address

Time

Victim L1 Cache

Spy L1 Cache

Shared L2 Cache (Inclusive)
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Attack Illustration

- Victim L1 Cache
- Spy L1 Cache
- Shared L2 Cache (Inclusive)

- Time
  - Evict
  - Wait

- Probe address
- Spy’s line
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Attack Illustration

- Victim L1 Cache
- Spy L1 Cache
- Shared L2 Cache (Inclusive)

Probes and Accesses:
- Green: Probe address
- Red: Spy's line

Time:
- Evict
- Wait
- Access
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Attack Illustration

- Victim L1 Cache
- Spy L1 Cache
- Shared L2 Cache (Inclusive)

- Probe address
- Spy’s line

- No Access
- Evict
- Wait

Timeline:
- Time
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Attack Illustration – Cont’d

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

Time
Cache Side-Channel Attack Classification

Eviction Strategies

- Conflict-based
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Cache Side-Channel Attack Classification

Eviction Strategies
- Conflict-based
- Flush-based

Evict
Wait
Analyze
Evict
Wait
Analyze

Evict

Victim L1 Cache
Spy L1 Cache

Shared L2 Cache

clflush addrX
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 has desirable characteristics
  – Prevents all known cache-based side channel attacks
  – Minimal performance overhead
  – No programmer intervention
  – Minor hardware modifications
SHARP: Preventing Inclusion Victims

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

Victim L1 Cache  Spy L1 Cache  Shared L2 Cache

Inclusion victim from other core is prevented

- Green: Probe address
- Red: Spy’s line
- White: 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

Inclusion victim from other core is prevented

Evict

Conflict

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
Obtaining Private Cache Information: **CVB**

- Using Core Valid Bits (CVB) only:
  - Step 1: Pick a line without any CVB set
  - Else Step 2: Pick a line with only the requester’s CVB set
  - Else Step 3: Random replacement + increment alarm counter

- Readily available
- Conservative
  - Silent evictions from private cache do not update CVB
Obtaining Private Cache Information: **Hybrid**

- Using combination of CVB and queries to private caches
  - Step 1: Pick a line without any CVB set
  - Else Step 2: Send queries to refresh CVB until a replacement candidate is found (all CVB bits zero, or only requester’s CVB set)
  - Else Step 3: Random replacement + increment alarm counter

✓ Accurate
✗ Significant network overhead to send messages
Obtaining Private Cache Information: SHARP

• Hybrid scheme with a limit on the query count
  – Step 1: Pick a line without any CVB set
  – Else Step 2: Send queries to refresh CVB for at most $N$ lines.
    If candidate is not found, use current CVB for the rest
    (to pick a line present only in the requester’s cache)
  – Else Step 3: Random replacement + increment alarm counter

✓ Accurate enough
✓ Low overhead
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

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 = sqr(r)
    if $e_i = 1$ then
        r = mul(r,b)
    end
end
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

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

- Baseline
- CVB
- Hybrid
- SHARP

Inability to evict shared data causes cache thrashing
“cvb” performs the worst
Secure Hierarchy-Aware Cache Replacement Policy (SHARP)

Modest slowdown of 6% due to large working set

Average execution time increase ≈ 1%

Execution Time

Baseline  CVB  Hybrid  SHARP

Normalized Execution Time
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