SCATTERCACHE: Thwarting Cache Attacks via Cache Set Randomization

Werner, Unterluggauer, Giner, Schwarz, Gruss, Mangard
August 15, 2019

Graz University of Technology
What is ScatterCache?

- Alternative design for n-way set associative caches
- Designed as countermeasures against cache attacks
  - Breaks the fixed link between addresses and cache sets
  - Increases the number of possible cache sets
  - IDs to change the mapping between security domains
    → Exploitation of side channel information is much harder
- Reuses established concepts
  - Skewed caches [Sez93]
  - Low latency cryptography (e.g., QARMA-64 [Ava17])
- Still similar to existing cache designs (usability, hardware)
Motivation and Background
CPU Cache

```c
printf("%d", i);
printf("%d", i);
```
generated using the CTA calibration tool [GSM15] on my i5-4200U laptop

Werner, Unterluggauer, Giner, Schwarz, Gruss, Mangard — Graz University of Technology
Regular 2-way Set Associative Cache

Memory Address

\[ f \]

\[ n \text{ bits} \quad b \text{ bits} \]

Cache Index

\[ 2^n \text{ cache sets} \]

Tag

\[ =? \]

\[ =? \]

\[ \text{Cache} \]

Way 1 Tag
Way 2 Tag

Way 1 Data
Way 2 Data

Data
Attacker Address Space

Cache

Victim Address Space

loads data

loads data

fast access

slow access
Why should we care?

- Cache attacks are powerful and break isolation boundaries
- Many attacking techniques
  - FLUSH+RELOAD, EVICT+RELOAD, FLUSH+FLUSH
  - PRIME+PROBE, EVICT+TIME
- Numerous attack scenarios
  - Extracting cryptographic keys
  - Keyloggers
  - Breaking of ASLR
  - Collection of private information
- Often used building block for further microarchitectural attacks
SCATTERCACHE
**ScatterCache - Idea**

<table>
<thead>
<tr>
<th>Set 0</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr. A</td>
<td></td>
<td>Addr. B</td>
<td></td>
</tr>
<tr>
<td>Addr. A</td>
<td></td>
<td>Addr. A</td>
<td></td>
</tr>
<tr>
<td>Addr. A</td>
<td></td>
<td>Addr. B</td>
<td></td>
</tr>
<tr>
<td>Addr. B</td>
<td></td>
<td>Addr. A</td>
<td></td>
</tr>
<tr>
<td>Addr. B</td>
<td></td>
<td>Addr. A</td>
<td></td>
</tr>
<tr>
<td>Addr. B</td>
<td></td>
<td>Addr. B</td>
<td></td>
</tr>
</tbody>
</table>

@DAC [Tri+18], @MICRO [Qur18]
How can we build such a ScatterCache?
Index Derivation Function (IDF) takes an address and returns a cache set.

- Depends on hardware key and optional Security Domain ID (SDID)
- Unique combination of cache lines for each address
  - Potential index collisions
  - One $n_{\text{ways}}$ multi-port memory

$\left( n_{\text{ways}} \cdot 2^{b_{\text{indices}}} + n_{\text{ways}} - 1 \right)$ possible cache sets

512 KiB (32 B lines), $n_{\text{ways}} = 8$, $b_{\text{indices}} = 11$

$\rightarrow 2^{96.7}$ sets
We want something that is closer to a traditional cache!

instead of this:

let’s do this:
ScatterCache - Concept

- Skewed cache [Sez93] (i.e., traditional cache with additional addressing logic) and an IDF
- Similar to building larger caches from smaller cache slices
- We use random replacement policy (for now)

$2^{b_{indices} \cdot n_{ways}}$ possible cache sets

512 KiB (32 B lines), $n_{ways} = 8$, $b_{indices} = 11$
→ $2^{88}$ sets
SCATTERCACHE - Selecting the IDF

- Inputs: cache line address, SDID, key
- Outputs: $n_{ways}$ indices with $b_{indices}$ bits
- Reuse concepts and existing cryptographic primitives
- SCv1: hashing variant
  - Block ciphers (e.g., PRINCE [Bor+12])
  - Tweakable block ciphers (e.g., QARMA [Ava17])
  - Permutation-based primitives (e.g., Keccak-p [Ber+11])
- SCv2: permutation variant
  - Prevents birthday-bound index collisions
  - No off-the-shelf primitives
System Integration
SCATTERCACHE as last level cache

Hardware managed key
- Randomly generated at boot time
- Rekeying with full cache flush
- Potential for iterative rekeying
  → concurrently developed CEASER-S @ISCA [Qur19]

SDID management via page table (indirection)
- x86: Page Attribute Tables (PATs)
- ARM: Memory Attribute Indirection Register (MAIRs)
ScatterCache requires no software support, default SDID = 0

But - OS support enables page-wise security domains
  → shared read-only pages can be private in the cache!

OS can define domains as needed
  (pages, processes, containers, VMs, . . .)

Software-based page “rekeying” by changing the SDID
Security and Evaluation
Applicable Cache Attacks

- **Unshared memory** has no shared (physical) addresses
  - No `FLUSH+RELOAD, EVICT+RELOAD, FLUSH+FLUSH`
  - Specialized `PRIME+PROBE` is possible

- **Shared, read-only memory**
  - Like unshared memory given OS support
  - Otherwise, eviction-based attacks are hindered

- **Shared, writable memory** can’t be separated
  - Eviction-based attacks are hindered
No end-to-end attack yet

- Simplified setting: perfect control, single access, no noise
- Investigate the building blocks in simulation and analytically

Finding congruent addresses ($n_{ways} = 8, b_{indices} = 11$)

- Full collisions are unlikely → use partial collisions
- Approach in the paper: $\approx 2^{25}$ profiled victim accesses
- Generalized by Purnal and Verbauwhede [PV19]: $\approx 2^{10}$

Evicting one set with 99% needs 275 addresses

Two PRIME+PROBE variants ($n_{ways} = 8, b_{indices} = 12$)

- 99% confidence: 35 to 152 victim accesses (repetitions)
- Between 9870 and 1216 congruent addresses

Investigate the effect of noise (coupon collector problem)
• Micro benchmarks using the gem5 full system simulator (ARM)
  • Poky Linux from Yocto 2.5 (kernel version 4.14.67)
  • GAP, MiBench, Imbench, scimark2
• SPEC CPU 2017 on custom cache simulator
• Cache hit rate always at or above levels of set-associative cache with random replacement
• Typically 2% – 4% below LRU on micro benchmarks, 0% – 2% for SPEC
Conclusion

- **SCATTERCACHE** builds upon skewed caches and low latency cryptographic primitives
  - Breaks the fixed link between addresses and cache sets
  - Removes the rigid assignment of cache lines to sets
  - Enables software control over the cache congruencies via SDIDs

- Comparable performance to contemporary caches
- Harder to attack even in very strong attack models
- Attacks are probabilistic and demand new approaches
- Still, more analysis is required in more realistic models to determine if and how often rekeying is needed
• the anonymous USENIX reviewers.
• our shepherd Yossi Oren.
• Antoon Purnal and Ingrid Verbauwhede from KU Leuven for their analysis.
• Our funding partners:
  • European Research Council (ERC)  
    Horizon 2020 grant agreement No 681402  
  • Intel
SCATTERCACHE: Thwarting Cache Attacks via Cache Set Randomization

Werner, Unterluggauer, Giner, Schwarz, Gruss, Mangard
August 15, 2019
Graz University of Technology
References


