ClepsydraCache – Preventing Cache Attacks with Time-Based Evictions

Jan Philipp Thoma¹, Christian Niesler², Dominic Funke¹, Gregor Leander¹, Pierre Mayr¹, Nils Pohl¹, Lucas Davi², Tim Güneysu¹

¹ – Ruhr-University Bochum, Bochum, Germany
² – University of Duisburg-Essen, Essen, Germany

Jan Philipp Thoma, M. Sc.
ReCap: Cache Randomization

- Cache randomization
  - Prevents efficient Prime + Probe attacks
  - Index is pseudorandomly generated from the address
  - Data is placed in one of the candidate entries
Prime + Prune + Probe Attack

Purnal et al., USENIX 2021

• Collect partially congruent addresses
  • Fill the cache with attacker controlled addresses

| Index | Way 0 | Way 1 | Way 2 | Way 3 |
|-------|-------|-------|-------|-------|
| 0     | Tag   | Data  | Tag   | Data  |
| 1     | Tag   | Data  | Tag   | Data  |
| 2     | Tag   | Data  | Tag   | Data  |
|       | ⋮     |       | ⋮     |       |
| 1021  | Tag   | Data  | Tag   | Data  |
| 1022  | Tag   | Data  | Tag   | Data  |
| 1023  | Tag   | Data  | Tag   | Data  |
Prime + Prune + Probe Attack
Purnal et al., USENIX 2021

- Collect partially congruent addresses
  - Fill the cache with attacker controlled addresses
  - Observe evictions and add colliding address to eviction set!

→ Cache attacks are harder, but still a threat!
ClepsydraCache

How can we prevent Prime+Prune+Probe?
A Clepsydra is an ancient time-measuring device worked by a flow of water.
How can we stop attackers from observing cache collisions?

- Prevent cache collisions!
- ClepsydraCache implements a combination of cache randomization and cache decay
- Efficient in hardware and small performance overhead
ClepsydraCache

... in a nutshell

- **Index randomization**
  - Prevent trivial eviction set construction
- Per-entry **time-to-live** (TTL)
  - The cache contains empty entries
  - Old data is evicted, recent data stays

![Diagram of Memory Address with Tag, Index, and F_K](image)

| Index | Way 0 | Way 1 | Way 2 | Way 3 |
|-------|-------|-------|-------|-------|
| 0     | Tag   | Data  | Tag   | Data  |
| 1     |       |       | Tag   | Data  |
| 2     | Tag   | Data  |       |       |
|       |       |       | Tag   | Data  |
| 1021  | Tag   | Data  |       |       |
| 1022  |       |       | Tag   | Data  |
| 1023  | Tag   | Data  |       |       |
ClepsydraCache

... in a nutshell

- Cache miss handling:
  - $F_K$ selects candidate entries
  - Fetch data from memory
  - Replacement policy prioritizes invalid entries
  - Set TTL to random value in range (min, max)
ClepsydraCache

... in a nutshell

- Cache hit handling:
  - $F_K$ selects candidate entries
  - Reset TTL to random value in range (min, max)
ClepsydraCache

... in a nutshell

- TTL expire:
  - Write-back if dirty
  - Invalidate entry

### Index Way 0 | Way 1 | Way 2 | Way 3
---|---|---|---
0 | Tag | Data | Tag | Data | Tag | Data | Tag | Data
1 | Tag | Data | Tag | Data | Tag | Data | - | -
2 | Tag | Data | - | - | Tag | Data | Tag | Data

...
Ideally, each randomized set contains one free entry!

- Monitoring not feasible in hardware
- Instead: Dynamic TTL scheduling
  - Adjust the speed with which the TTL expires globally
  - On conflicts: expire other entries faster
  - Otherwise, slowly allow longer lifetimes
Hardware Support

How do we implement this?

- **Option 1)** Counters in every cache set
  - Expensive to maintain
  - Could make use of in-memory computation

- **Option 2)** Use a capacitor to store the TTL
  - Can be placed on top of a memory cell
  - Does not compete for silicon area
Security
Prime + Prune + Probe

- The attacker needs to prime the cache
  - High activity will cause other entries to be evicted
  - Attacker needs to fill all candidate entries
  - Cannot distinguish between conflict- and TTL-based evictions

![Diagram of Memory Address and Cache Index]

| Index | Way 0 | Way 1 | Way 2 | Way 3 |
|-------|-------|-------|-------|-------|
| 0    | Tag Data | Tag Data | Tag Data | | |
| 1    | Tag Data | Tag Data | | Tag Data |
| 2    | Tag Data | | Tag Data | |
| 1021 | Tag Data | Tag Data | Tag Data | Tag Data |
| 1022 | | Tag Data | Tag Data | Tag Data |
| 1023 | | Tag Data | Tag Data | Tag Data |
Performance

Cache Conflicts

• Evaluation using gem5
  • Eviction set construction was not possible
  • Reduced conflict-based evictions by over 90% (Parsec and SPEC CPU 2017)
Performance

Miss Rate

Cache Miss Rate

- Miss Rate
- Cache Miss Rate
- bodytrack
- freqmine
- blackscholes
- caen
- ferret
- raytrace
- decc
- dup
- swaptions
- fatesim
- fluidanimate
- wips
- streamcluster
- x264
Performance
Total Overhead

- Total overhead between **0.37%** and **5.25%**
- Performance increase results from access patterns in non-randomized setting
