SCAR: Selective Cache Address Remapping for Mitigating Cache Side-Channel Attacks

Pavitra Prakash Bhade[1], Olivier Sentieys[2], Sharad Sinha[1]

[1]School of Mathematics and Computer Science, Indian Institute of Technology Goa, India
[2]University of Rennes, INRIA

Abstract

Cache side channel attacks (CSCA) that exploit cache conflicts pose a significant threat to the security of shared caches. To counter these attacks, cache designs incorporating cache randomization have emerged as a highly effective solution. However, these techniques rely on address remapping for all instructions, thus introducing performance drawbacks and risks of breaking remapping within the rekeying interval. We propose a novel cache architecture and control mechanism called SCAR that performs selective remapping of instructions, thus enhancing security while minimizing performance overhead. SCAR uses modified cache search and replacement algorithms, along with minimal addition to the cache hardware architecture.

Introduction

Cache mappings and replacement policies help to effectively use the limited size caches through sharing of cache set/lines. This introduces "conflict-based attacks" as some addresses must share the same cache set/line. Most existing methods involve remappings for all instruction addresses, often resulting in performance overhead. Furthermore, it is still prone to predictive analysis-based key extraction, though frequent *rekeying* helps to some extent. In response to these challenges, we propose a novel cache architecture combined with a selective randomization technique that remaps only the addresses of instructions that conflict with secret-dependent instructions and whose execution trace can reveal the secret. Our approach reduces the overall performance overhead and introduces additional entropy, thereby fortifying the system against attacks.

Methodologies

We propose modifications in the cache structure as well as in the cache search and replacement policy.

1. **Update in cache structure:** We introduce two new bits, S (Secret) and R (Remap), in each cache line. The S bit indicates whether the mapped line includes secret information, and the R bit indicates whether this line is loaded in the cache after remapping the address.
2. **Modification in cache search and replacement algorithms** Algorithm 1 outlines the method for searching an instruction/data (henceforth referred to as an element) in the cache memory. This process involves two searches in the cache. Hardware optimizations can be applied to improve performance. Algorithm 2 describes the cache replacement process. To enhance security, the remapping key is regenerated after every 100 accesses to the secret cache lines [1], invalidating only the secret and remapped locations, unlike full cache invalidation in existing randomization methods.

Algorithm 1 Search element in cache using the SCAR technique

```
1:  procedure CACHE SEARCH
2:      if Addr "A" present in cache then
3:          if Remap bit ==0 then
4:              Cache Hit
5:          end if
6:      else
7:          Remaps "A" to "X"
8:          if Addr "X" present in cache then
9:              if Remap bit ==1 then
10:                 Cache Hit
11:             end if(end all ifs and procedure)
```

Experiments and Results

We have implemented SCAR in the Comet RISC V core [2]. For our experiments, we use ZephyrOS, an open-source operating system dedicated to embedded systems. We performed tests on the RSA 512 encryption algorithm along with the Mibench benchmark suite to emulate some workloads. To mark the secret [3], we annotate the respective section of the code. We launched eviction-based attacks like Prime+Probe and Evict+Probe to test our method. We observed that the secret section is not evicted by the launched attacks on our proposed cache structure, thus showing

Algorithm 2 Replace element in cache using the SCAR technique

```
1:  procedure CACHE REPLACEMENT POLICY
2:      if Address "A" has to evict Address "B" then
3:          if Addr "A" is a secret then
4:              Remap "A" to "X"
5:              Search replacement location for "X"
6:              Replace at found location
7:              Set Secret Bit of Addr "X" to 1
8:              Set Remap Bit of Addr "X" to 1
9:          else
10:             if Secret Bit of Addr "B" ==0 then
11:                 Replace
12:             else
13:                 Remap Addr "A" to "X"
14:                 Search replacement location for "X"
15:                 Replace at the found location
16:                 Set Remap Bit of "X" to 1
17:             end if(end all ifs and procedure)
```

successful mitigation of attacks in our experiments.

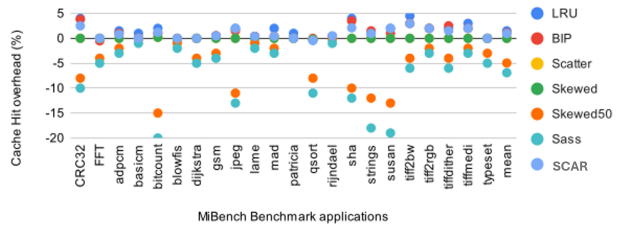

Figure 1: *Cache hit comparison with [4] on a 32KB cache for Mibench*

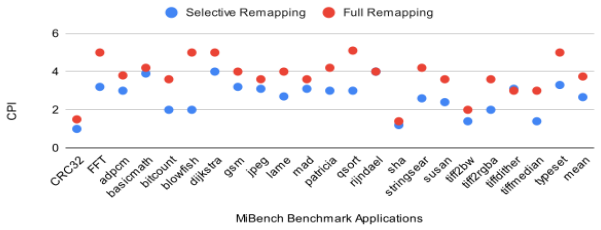

Figure 2: *Comparison of clock cycles per instruction (CPI) in selective vs full remapping method*

Fig. 1 compares our work with the current state-of-the-art techniques, as analysed in [4], for a 32KB cache with Mibench benchmarks as the application. These results demonstrate that SCAR achieves a higher cache hit ratio than the other techniques. Figure 2 shows the impact on clock cycles per instruction (CPI) due to SCAR against the full remapping methods for the same benchmark applications considered. The CPI os SCAR is around 1.18 clock cycles lower than full address remapping, thus leading to much less impact on performance. Fig. 3 depicts the overhead in execution time due to introducing remap logic in SCAR over LRU-based cache design.

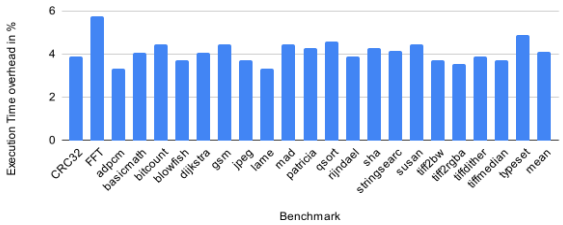

Figure 3: *Overhead in execution time due to remapping in SCAR over LRU caches*

We have also performed experiments and analysed the impact on cache hits, misses, remaps, and performance overhead by varying the cache size and number of secrets monitored. However, due to page length limitations, we have not included these results.

In applications with no secrets to be monitored, SCAR will default to traditional cache search and replacement, i.e., without any remapping. This protects the original performance without any impact compared to the current remap techniques, where encryptions are bound to happen.

Conclusion

Targeting RISC-V cores, we have proposed a minimal enhancement in the cache microarchitecture and in the search and re-placement policy to mitigate conflict-based CSCA. We have analysed the performance overhead of our technique with state-of-the-art cache randomisation approaches. Future work includes synthesizing the proposed cache structure in various RISC-V microarchitectures to gain deeper insights on security and performance evaluations.

References

[1] Moinuddin K. Qureshi. "CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping". In: *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2018, pp. 775–787. DOI: 10.1109/MICRO.2018.00068.

[2] Simon Rokicki et al. "What You Simulate Is What You Synthesize: Designing a Processor Core from C++ Specifications". In: *Proceedings of the International Conference on Computer-Aided Design, ICCAD 2019, Westminster, CO, USA, November 4-7, 2019*. Ed. by David Z. Pan. ACM, 2019, pp. 1–8. ISBN: 9781728123509.

[3] Michael Schwarz et al. "ConTExT: A Generic Approach for Mitigating Spectre". In: *27th Annual Network and Distributed System Security Symposium (NDSS)*. 2020.

[4] Lukas Giner et al. "Scatter and Split Securely: Defeating Cache Contention and Occupancy Attacks". In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023, pp. 2273–2287. DOI: 10.1109/SP46215.2023.10179440.