Side-Channel Security

Chapter 8: Side-Channel Attacks on The OS Page Cache

Sudheendra Neela

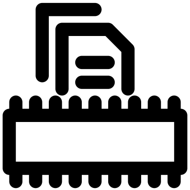
April 3, 2025

Graz University of Technology

Motivation

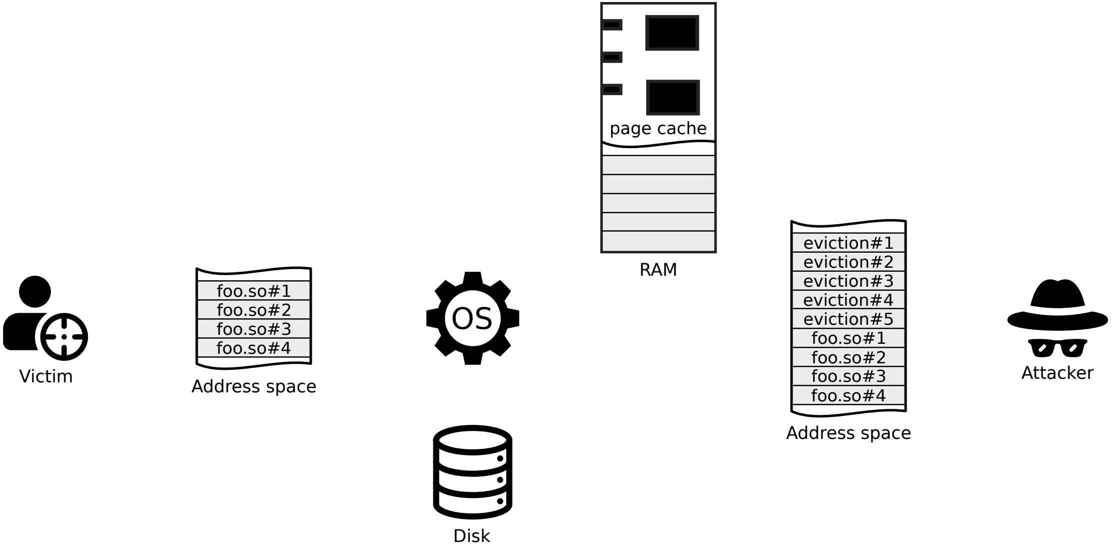
- So you know side-channel attacks on the CPU Cache:
  - Flush+Reload
  - Flush+Flush
  - Prime+Probe
- Why do these attacks work?
- Attacker and victim: share the CPU cache

The Operating System's Page Cache

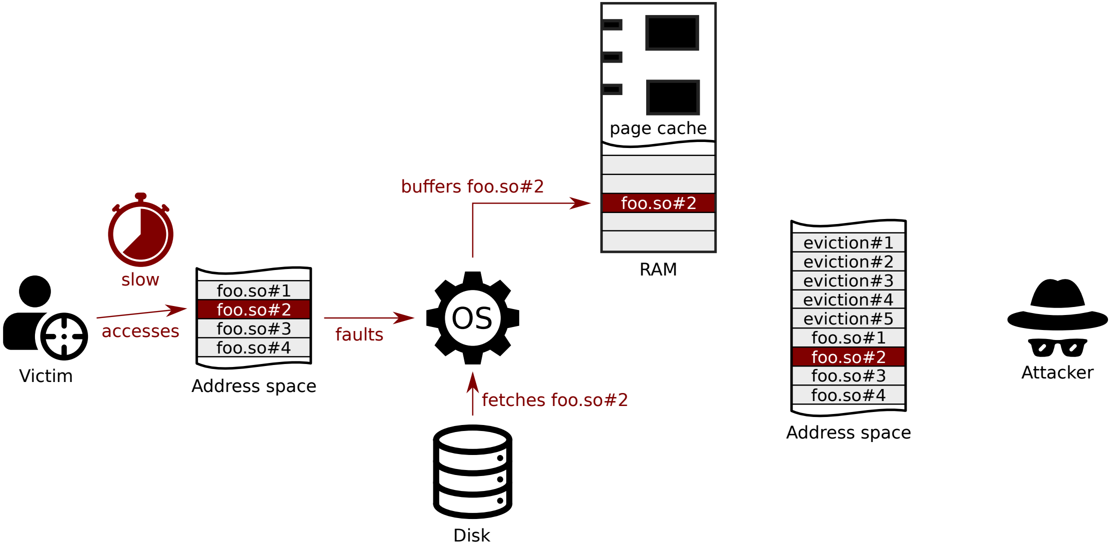


- Managed by operating system
- Buffers pages in RAM for faster accesses
- Behaves as a fully-associative cache
- State of pages is tracked:
  - No write access → clean → no write back
  - Write access → dirty → write back
- Implemented by all major operating systems

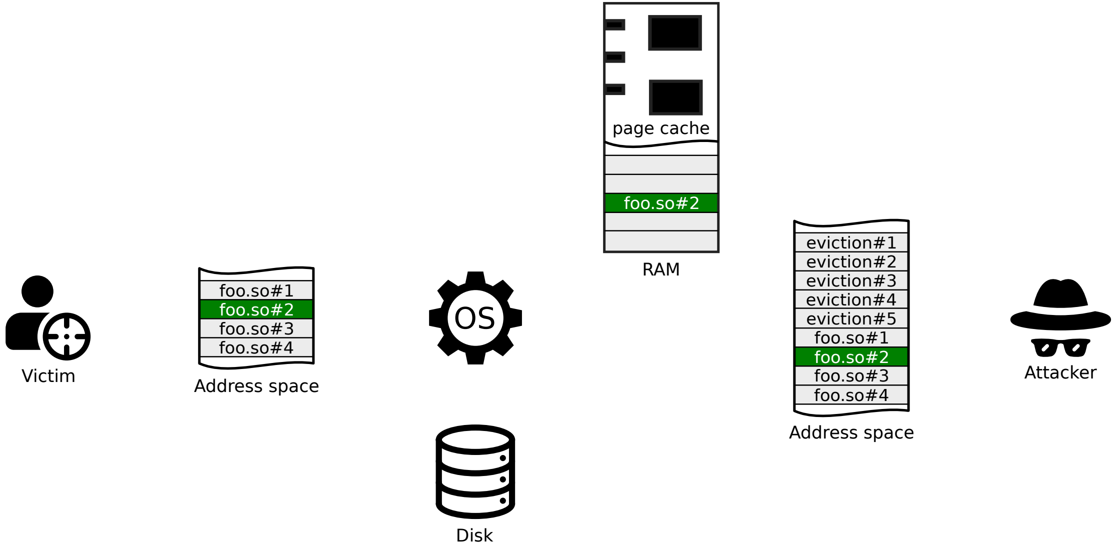
Page Cache Attacks [4]



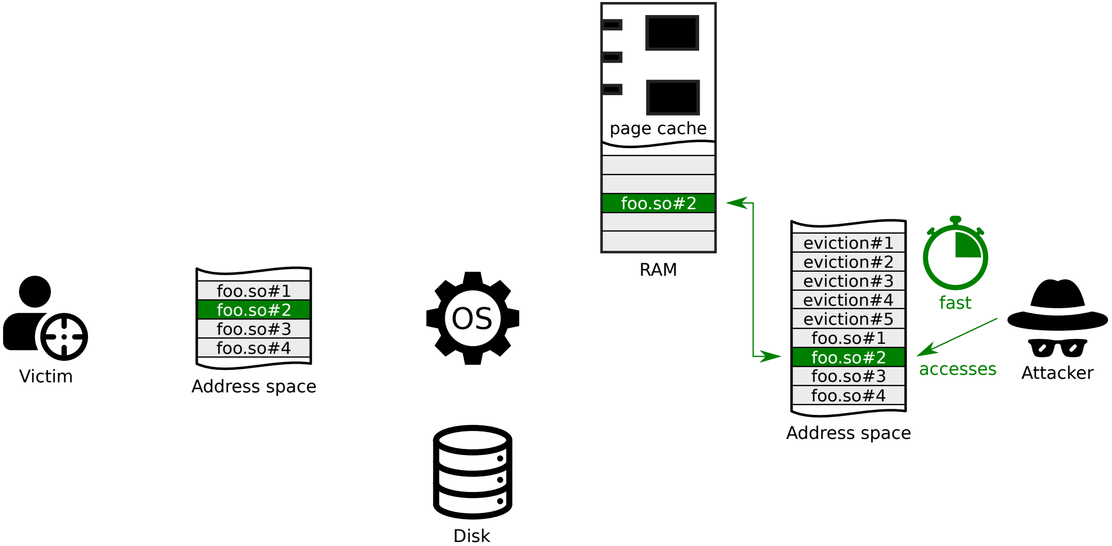
Page Cache Attacks [4]



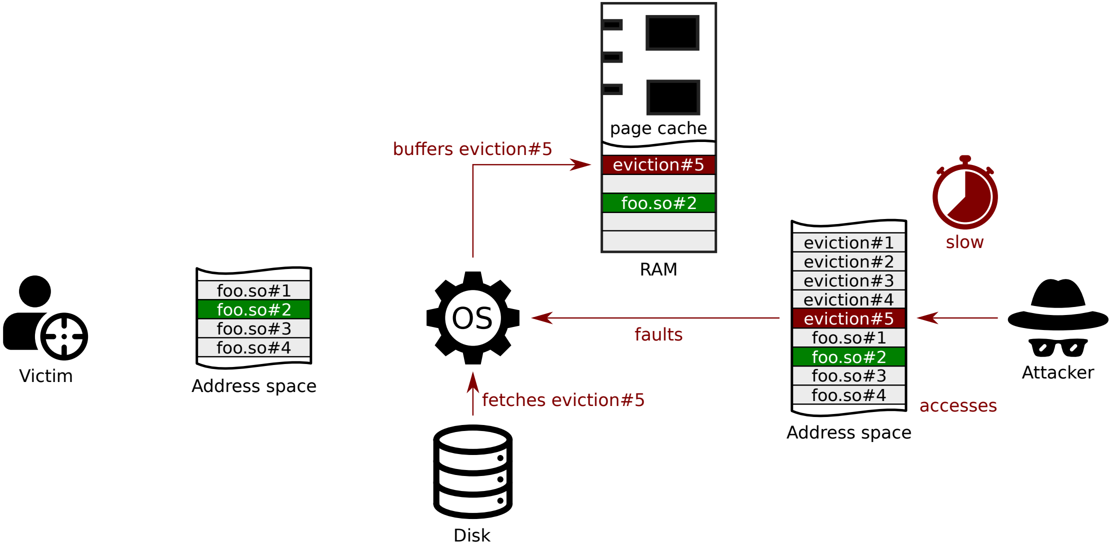
Page Cache Attacks [4]



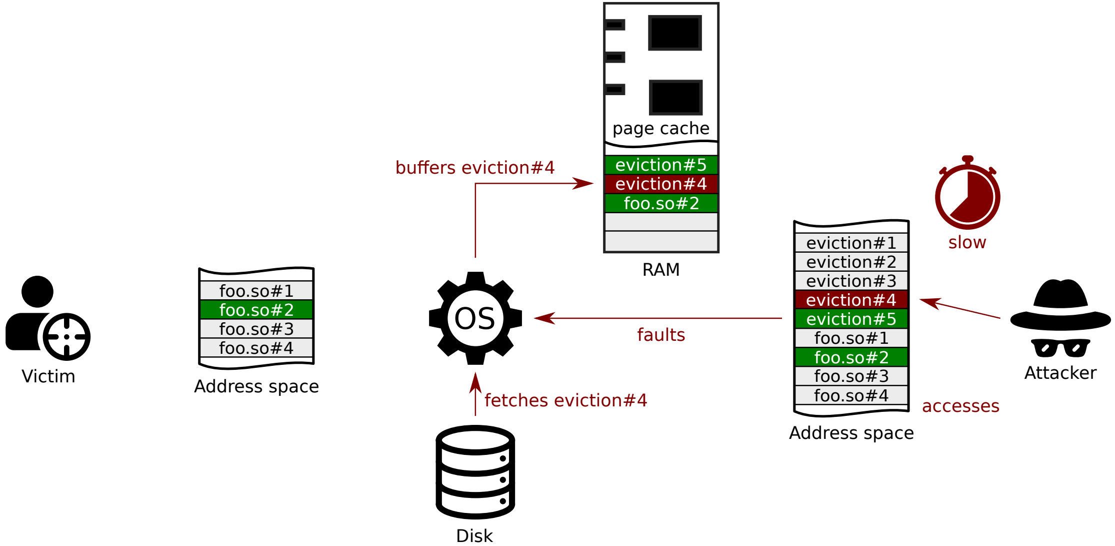
Page Cache Attacks [4]



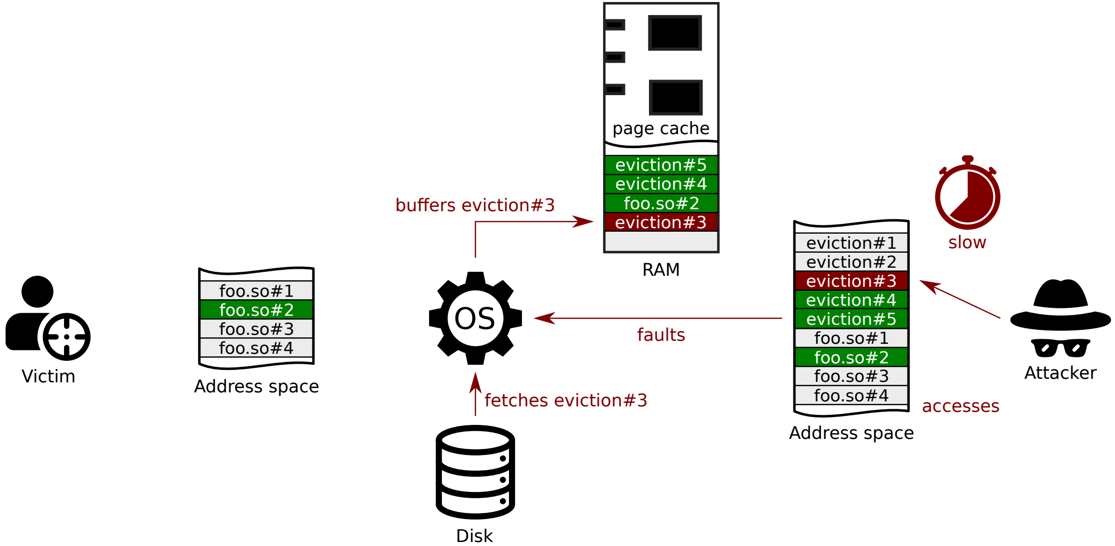
Page Cache Attacks [4]



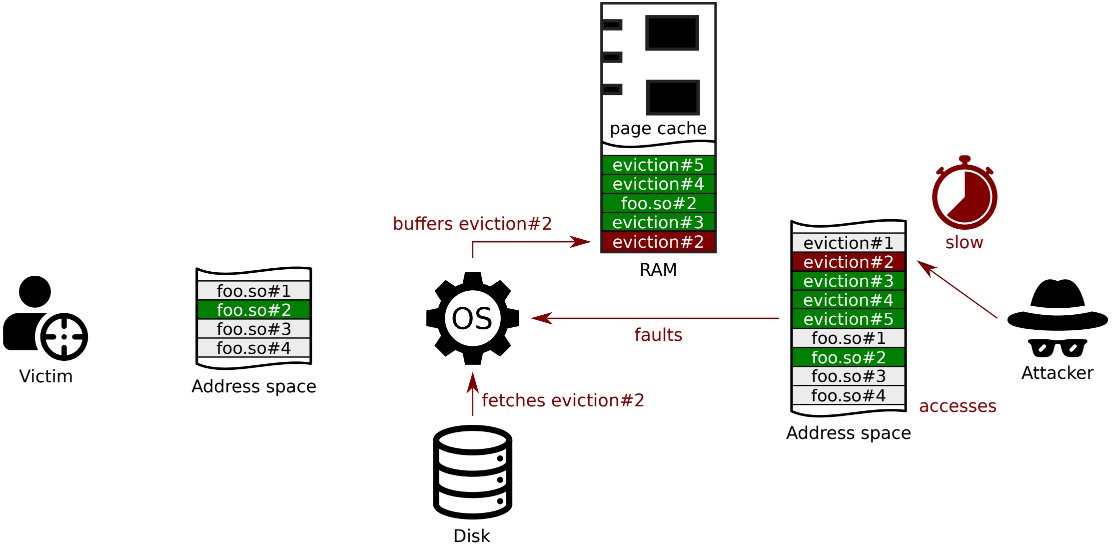
Page Cache Attacks [4]



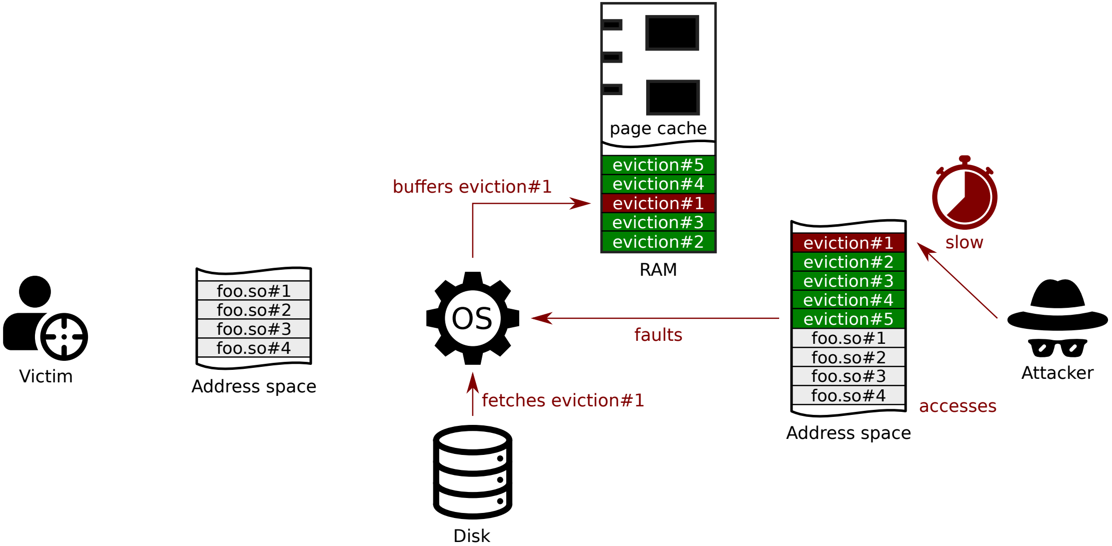
Page Cache Attacks [4]



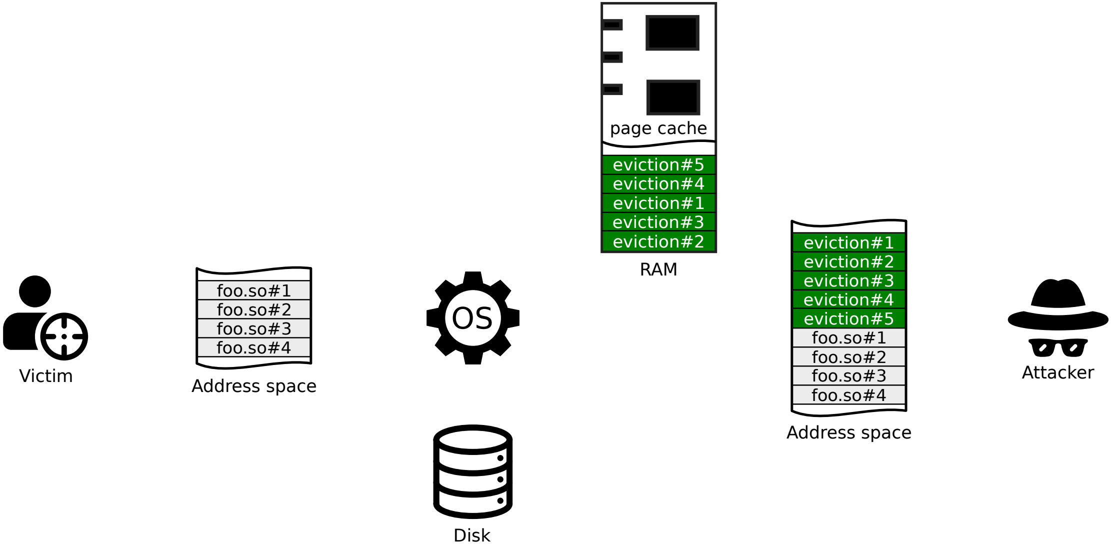
Page Cache Attacks [4]



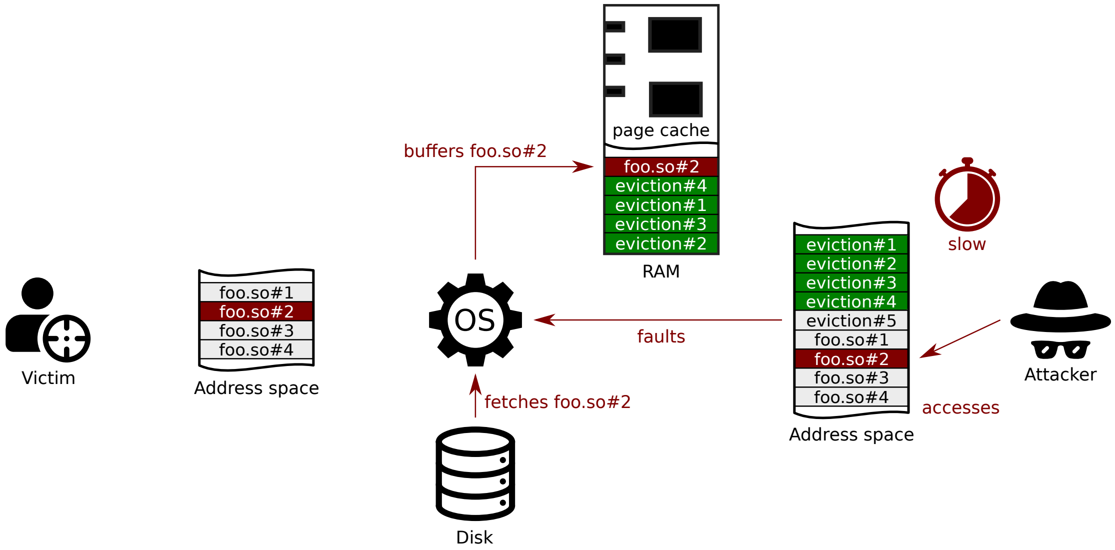
Page Cache Attacks [4]



Page Cache Attacks [4]

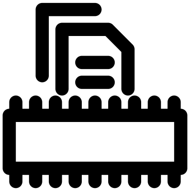


Page Cache Attacks [4]



You don't even need to time it! [4]

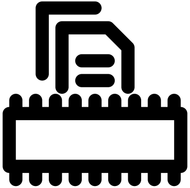
`mincore` ($2.04\,\mu$s) — Linux syscall



- Takes virtual memory range, returns vector (man 2 mincore), introduced in 2000 [5]
- Indicates presence of queried pages in page cache

```
MINCORE(2)              Linux Programmer's Manual              MINCORE(2)

NAME
       mincore - determine whether pages are resident in memory

SYNOPSIS
       #include <unistd.h>
       #include <sys/mman.h>

       int mincore(void *addr, size_t length, unsigned char *vec);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       mincore():
           Since glibc 2.19:
               _DEFAULT_SOURCE
           Glibc 2.19 and earlier:
               _BSD_SOURCE || _SVID_SOURCE

DESCRIPTION
       mincore()  returns a vector that indicates whether pages of the calling
       process's virtual memory are resident in core (RAM), and  so  will  not
       cause  a  disk  access  (page fault) if referenced.  The kernel returns
       residency information about the pages starting at the address addr, and
       continuing for length bytes.
```

You don't even need to time it! [4]



`QueryWorkingSetEx` (465.91 ns) — Windows API

- Takes process handle + virtual memory address, returns struct
- Exposes attributes of queried page ...
- ... presence in working set
- ... number of working sets containing page (`ShareCount`)

Attacks on Firefox [4]

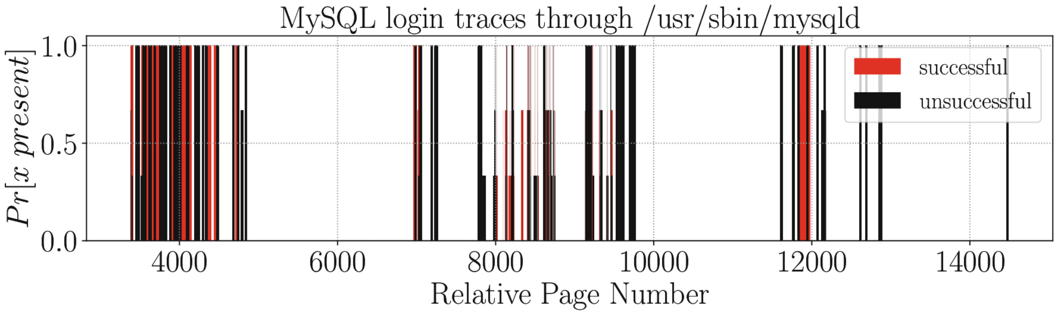
- Threat model: cross-user
- Executable and shared libraries: shared across users
- Firefox executable: 1.1 MB; ~280 pages:
  - Pages 54, 55: New browser window
- libmozavcodec.so: 3.8 MB; ~975 pages:
  - Pages 0 to 416: YouTube (streaming `webm`)
  - Pages 64 to 80, 240 to 256: ? (streaming `mp4`)
- libmozavutil.so: 604 KB; ~150 pages:
  - Pages 0 to 23: Media sites — Facebook, YouTube
- libxul.so: 146 MB! ~37 350 pages!

Limitations

- Eviction is not easy
- Fully-associative cache: memory should be completely filled before eviction starts
- Try to reduce free memory in system with `malloc` and `mlock`
- Linux eviction: 149 ms
- Kernel may read 32 pages ahead for optimization — read-ahead mechanism: more noisy for timing-based tests
- `mincore` syscall mitigated in 2019 [2, 3]

Attacks Across Docker Containers [1]

- Threat model: cross-container
- Containers use `OverlayFS` (Union Mount FS): shared files
- Use `mincore` [4] to determine page cache residency
- Detect successful login attempts into MySql



Another vector to Leak Page Cache Residence [6] (2023)

- `preadv2` syscall: "read data into multiple buffers"
- `RWF_NOWAIT` flag: "Do not wait for data which is not immediately available"
- Overcoming read-ahead: `madvise` with `MADV_RANDOM` flag to "expect page references in random order"
- Chrome browser (2017-2022): information of key events ("KeyA", "KeyB") in different pages due to linker optimizations
- Evict target file, use preadv2 to determine keypress

```
DOM_CODE(0x070004, 0x001e, 0x0026, 0x001e, 0x0000, "KeyA", US_A), // aA
DOM_CODE(0x070005, 0x0030, 0x0038, 0x0030, 0x000b, "KeyB", US_B), // bB
DOM_CODE(0x070006, 0x002e, 0x0036, 0x002e, 0x0008, "KeyC", US_C), // cC
```

Side-Channel Security

Chapter 8: Side-Channel Attacks on The OS Page Cache

Sudheendra Neela

April 3, 2025

Graz University of Technology

References

[1] Boskov, N., Radami, N., Tiwari, T., and Trachtenberg, A. (2022). Union Buster: A Cross-Container Covert-Channel Exploiting Union Mounting. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*.

[2] Corbet, J. (2019a). Defending against page-cache attacks.

[3] Corbet, J. (2019b). Fixing page-cache side channels, second attempt.

[4] Gruss, D., Kraft, E., Tiwari, T., Schwarz, M., Trachtenberg, A., Hennessey, J., Ionescu, A., and Fogh, A. (2019). Page cache attacks. In *CCS*.

[5] Lever, C. (2000). [patch] madvise() against 2.3.52-3.

[6] Schwarzl, M., Kraft, E., and Gruss, D. (2023). Layered Binary Templating. In *ACNS*.