LLCWhisp: Feasibility of Occupancy Attacks on Fully Associative LLCs with Random Eviction

Abhishek Khandelwal - BT/CSE/220040
Pallav Goyal - BT/CSE/220747

Supervisors:
Prof. Debadatta Mishra
Prof. Mainak Chaudhuri
Ms. Yashika Verma (Mentor)

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur

Background & Motivation

- **Last-Level Cache (LLC)** is commonly shared across cores in multi-core systems.
- Such shared caches are vulnerable to **timing and conflict-based side-channel attacks**.

Background & Motivation

- **Last-Level Cache (LLC)** is commonly shared across cores in multi-core systems.
- Such shared caches are vulnerable to **timing and conflict-based side-channel attacks**.
- Fully Associative LLCs with **Random Eviction Policy (RFA-LLC)** is a proposed mitigation for conflict-based side-channel attacks.
- But **RFA-LLC** is vulnerable to **occupancy-based attacks**.

Background & Motivation

- These attacks use:
  - **Cache fills**
  - **Difference in hit and miss latency**
  - **Disturbance measurements due to different Eviction patterns**

Problem Statement

- Most existing evaluations are performed in **simplified setups** without **OS noise**.

Problem Statement

- Most existing evaluations are performed in **simplified setups** without **OS noise**.
- **Real-world systems** include:
  - **OS scheduling noise**
  - **Interrupts and background tasks**
- These introduce significant **noise and uncertainty** in attack performance.

Objective

- Evaluate the feasibility of **occupancy-based attacks** on RFA-LLCs in real system setting
  - In the presence of realistic OS and system-level interference
  - Using **gem5 full-system simulation** with Linux OS

Occupancy-Based Attack Model

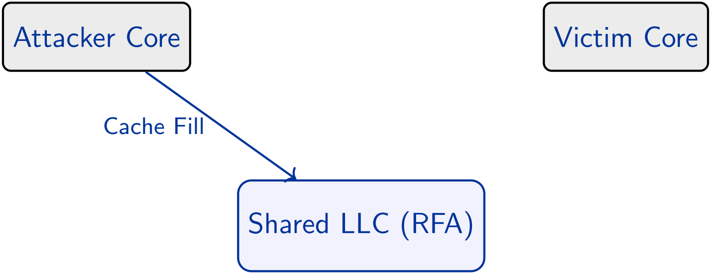
Attacker Core

Occupancy-Based Attack Model

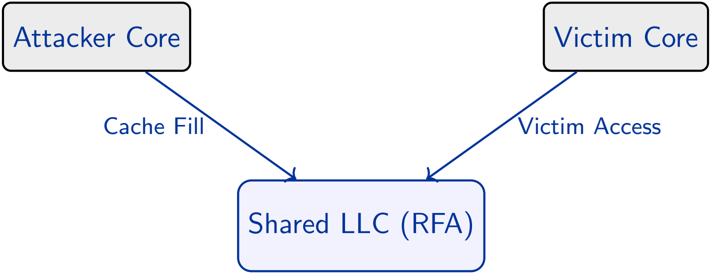
Attacker Core

Victim Core

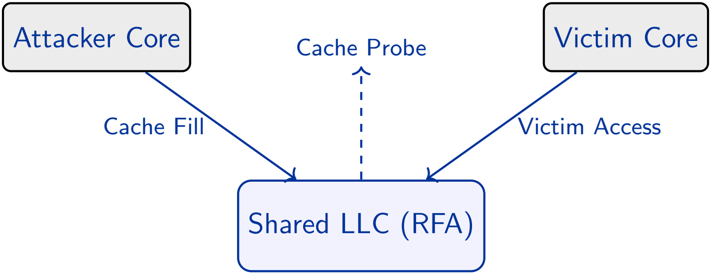
Occupancy-Based Attack Model



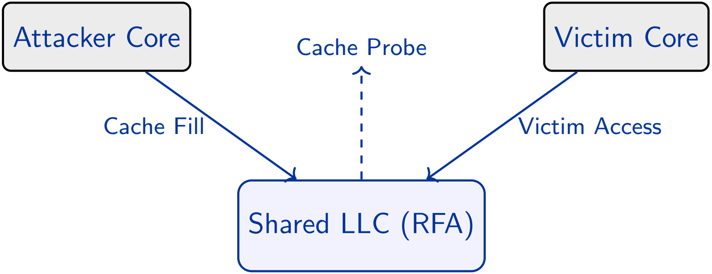
Occupancy-Based Attack Model



Occupancy-Based Attack Model



Occupancy-Based Attack Model



Attack Steps:
1. Attacker fills the LLC
2. Victim evicts some cache lines
3. Attacker probes occupancy
4. Inference via disturbance

Project Contributions

- Implementation and experimentation of FA cache with random replacement policy in gem5
- Validation of cache fill and probe measures for occupancy-based attacks in realistic setting
- Analyzing the effect of OS noise on occupancy-based attack performance

Project Contributions

- Implementation and experimentation of FA cache with random replacement policy in gem5
- Validation of cache fill and probe measures for occupancy-based attacks in realistic setting
- Analyzing the effect of OS noise on occupancy-based attack performance

Experimental Setup and Implementation

- **Simulator:** Experiments are conducted on the **gem5 full-system simulator** using the **Linux operating system**.
- **System Noise:** All experiments are conducted in presence of **realistic OS scheduling and background activity**, to evaluate feasibility of occupancy attacks in realistic settings.

- **Fully-Associative LLC:**

$$\text{Set Associativity} = \frac{\text{Cache\_Size}}{\text{Block\_Size}}$$

so that the LLC becomes a single-set (fully associative) cache.

FA-LLC Configuration & Random Replacement

- **Fully-Associative LLC:**

$$\text{Set Associativity} = \frac{\text{Cache\_Size}}{\text{Block\_Size}}$$

  so that the LLC becomes a single-set (fully associative) cache.

- **Efficient Random Eviction:** In gem5 we maintain an **Invalid Blocks List** data structure that:
  - Tracks all cache lines invalidated
  - Allows constant-time insertion/removal as blocks become invalid/valid

Gem5 configurations

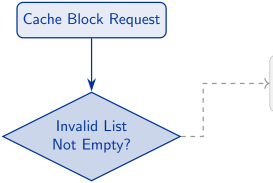
ISA	x86
Frequency	3GHz
Cores	2
L1-I, L1-D	32KB, 8-way, 64B line, 1 cycles, LRU, no prefetch
L2	256KB, 16-way, 64B line, 3 cycles, no prefetch, LRU
L3 (LLC)	2MB, 32768-way, 64B line, 15 cycles, no prefetch, RandomRP
DRAM	SingleChannelDDR3_1600 3GB

Table: Gem5 System Configurations

Cache Block Request Handling in gem5

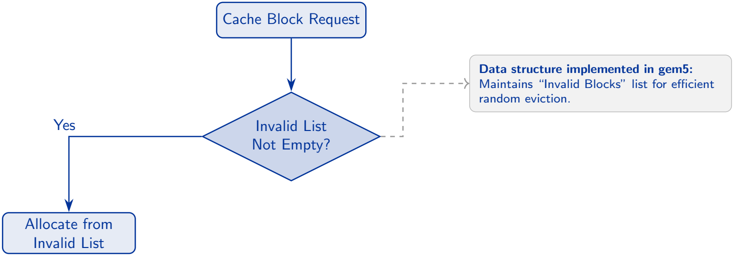
Cache Block Request

Cache Block Request Handling in gem5

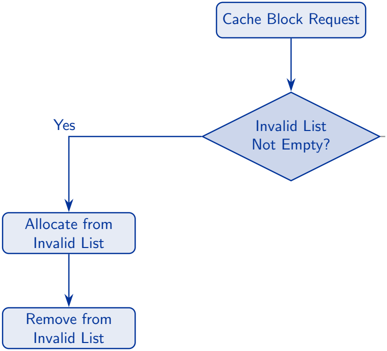


Data structure implemented in gem5:
Maintains "Invalid Blocks" list for efficient random eviction.

Cache Block Request Handling in gem5

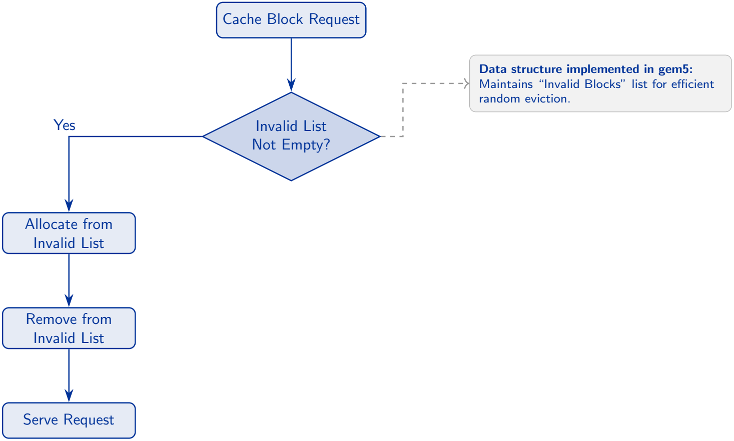


Cache Block Request Handling in gem5

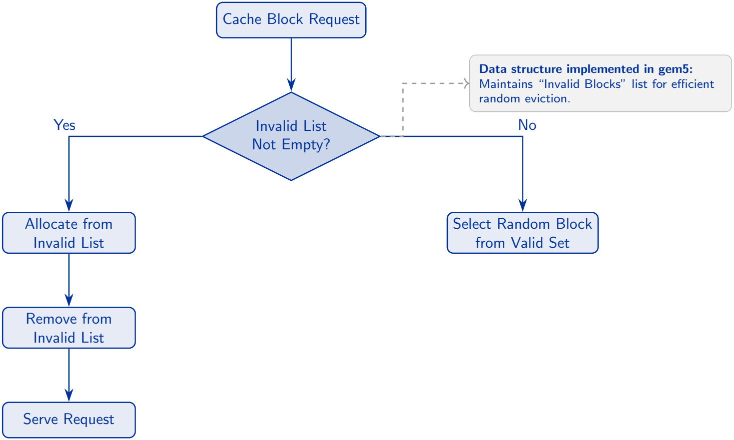


Data structure implemented in gem5:
Maintains "Invalid Blocks" list for efficient random eviction.

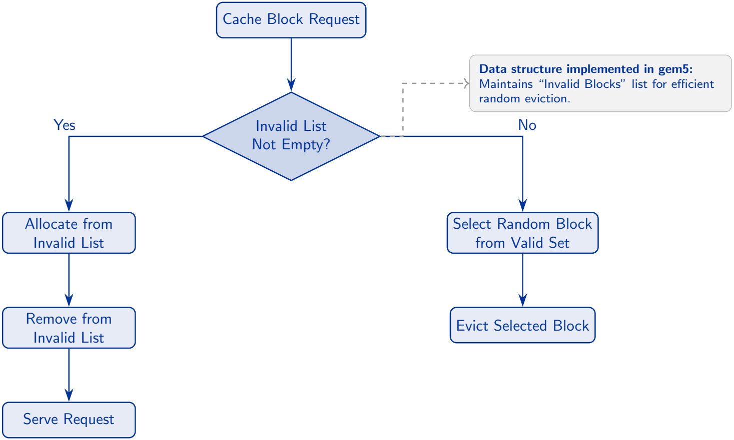
Cache Block Request Handling in gem5



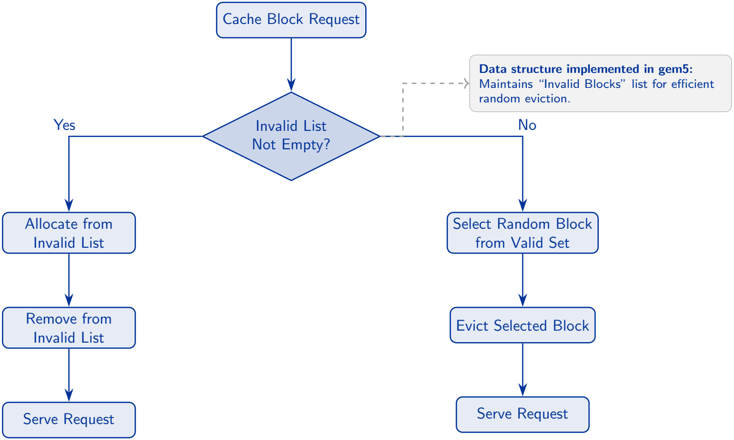
Cache Block Request Handling in gem5



Cache Block Request Handling in gem5



Cache Block Request Handling in gem5



Project Contributions

- Implementation and experimentation of FA cache with random replacement policy in gem5
- Validation of cache fill and probe measures for occupancy-based attacks in realistic setting
- Analyzing the effect of OS noise on occupancy-based attack performance

CRFill and CRProbe: Overview

- **CRFill:** Fill the cache by accessing our data to occupy significant portion of LLC.

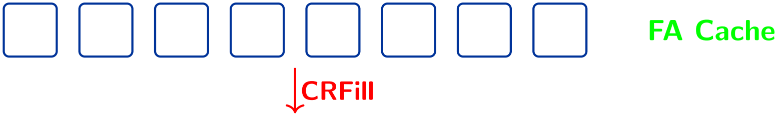
CRFill and CRProbe: Overview

- **CRFill:** Fill the cache by accessing our data to occupy significant portion of LLC.
- **CRProbe:** Accessing data and using access latency to detect hit or miss which helps calculating occupancy.

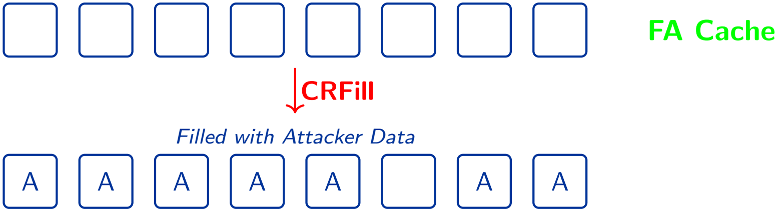
CRFill and CRProbe Visualization

FA Cache

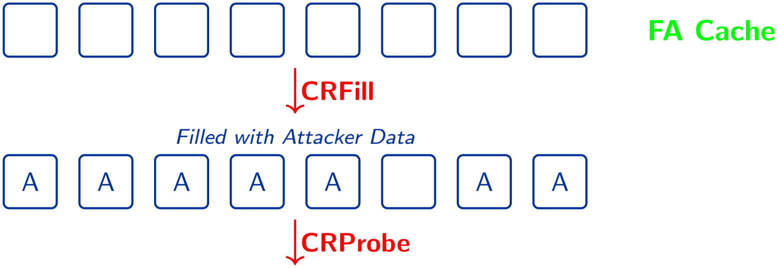
CRFill and CRProbe Visualization



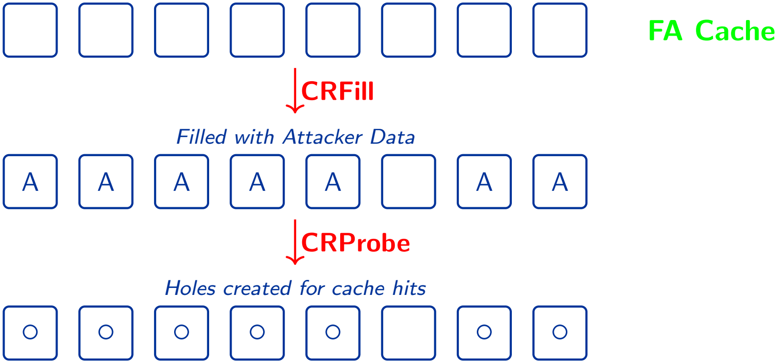
CRFill and CRProbe Visualization



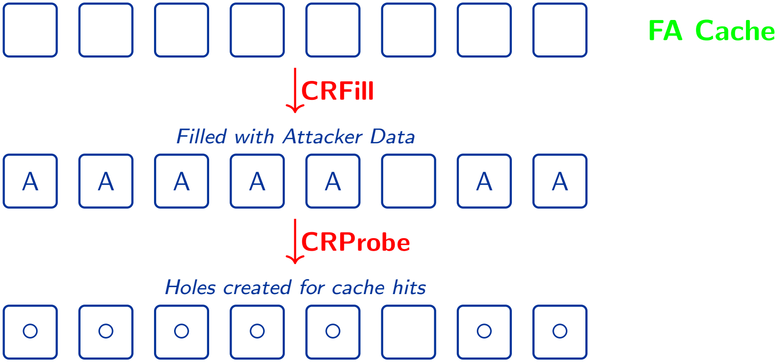
CRFill and CRProbe Visualization



CRFill and CRProbe Visualization



CRFill and CRProbe Visualization



CRProbe Pseudocode

CRProbe Algorithm

```
int i = arr_size - 1
while i ≥ 0 do
    t1 = rdtsc
    load(arr[i])
    delay = rdtsc - t1
    clflush(arr[i])
    if delay < hit_miss_threshold OR i == arr_size-1
then
        occ++
    i = i - 1
done
```

CRProbe Pseudocode

CRProbe Algorithm

```
int i = arr_size - 1
while i ≥ 0 do
    t1 = rdtsc
    load(arr[i])
    delay = rdtsc - t1
    clflush(arr[i])
    if delay < hit_miss_threshold OR i == arr_size-1
then
        occ++
    i = i - 1
done
```

Note: Misses are handled carefully to avoid evicting filled lines—crucial in random eviction LLCs.

Evaluating Cache Fill Effectiveness

- Goal: To study the effectiveness of cache filling under realistic OS scheduling and background activity.
- Setup: Multiple access patterns tested with data sizes and access counts denoted as x^a — where data size is x times of LLC size and a is the number of iterations over the data..

Evaluating Cache Fill Effectiveness

- Goal: To study the effectiveness of cache filling under realistic OS scheduling and background activity.
- Setup: Multiple access patterns tested with data sizes and access counts denoted as x^a — where data size is x times of LLC size and a is the number of iterations over the data..
- Metric: **Achievable Occupancy** of the fully associative LLC after fill.
- All experiments were conducted on **gem5 full-system simulator** running **Linux OS**.

Evaluating Cache Fill Effectiveness

- Goal: To study the effectiveness of cache filling under realistic OS scheduling and background activity.
- Setup: Multiple access patterns tested with data sizes and access counts denoted as x^a — where data size is x times of LLC size and a is the number of iterations over the data..
- Metric: **Achievable Occupancy** of the fully associative LLC after fill.
- All experiments were conducted on **gem5 full-system simulator** running **Linux OS**.

Occupancy values reflect the ability of attacker to maintain control over cache blocks despite system noise.

Static CRFill

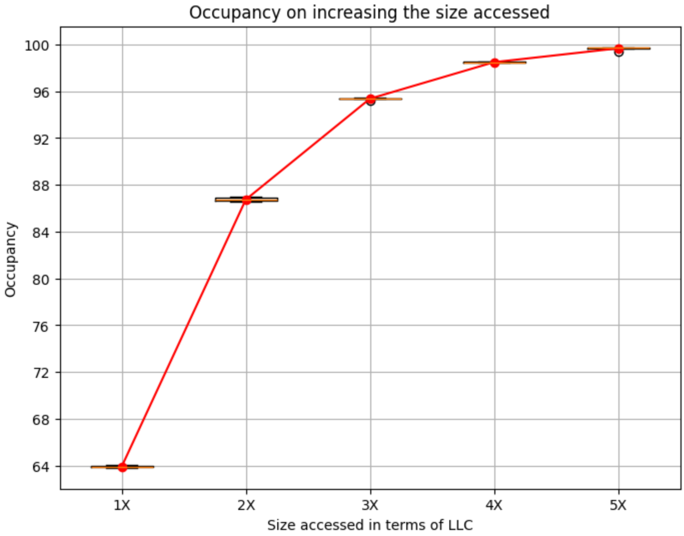


Figure: CRFill step

Static CRFill

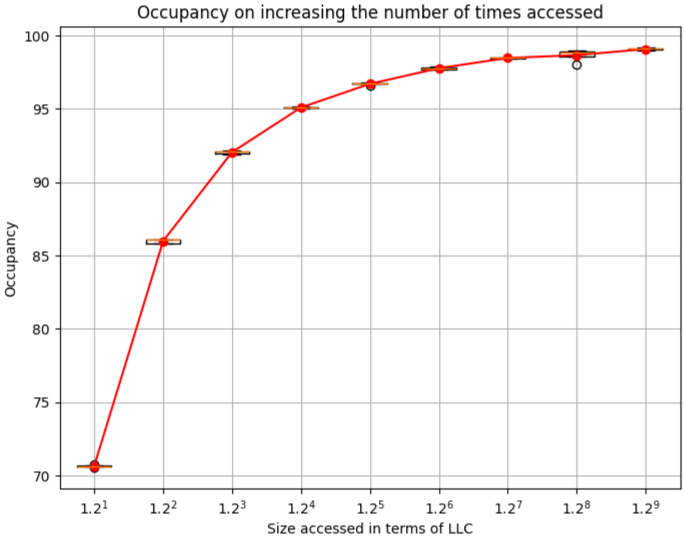


Figure: CRFill step

Dynamic CRFill

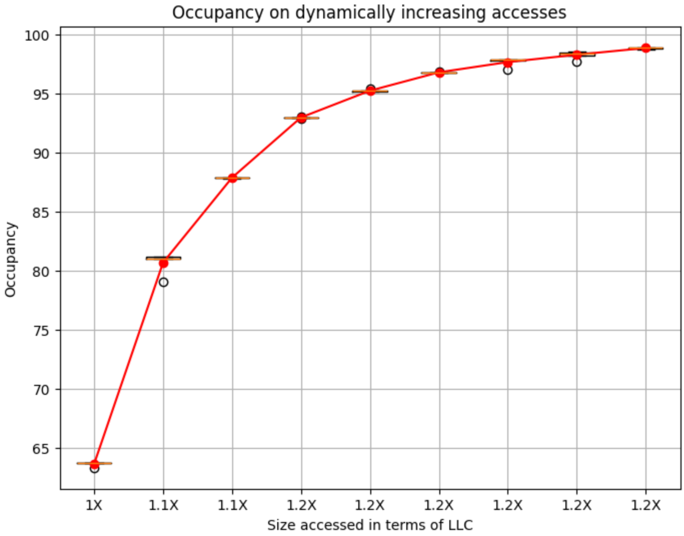


Figure: CRFill step

Key Observations

Inference

Occupancy-based attacks are practically feasible even in noisy, full-system environments. Minimal OS-induced disturbance observed, with attackers consistently achieving high occupancy across patterns.

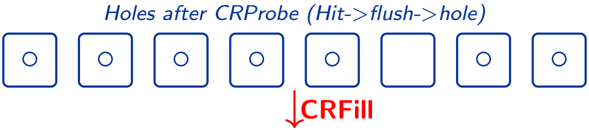
Project Contributions

- Implementation and experimentation of FA cache with random replacement policy in gem5
- Validation of cache fill and probe measures for occupancy-based attacks in realistic setting
- Analyzing the effect of OS noise on occupancy-based attack performance

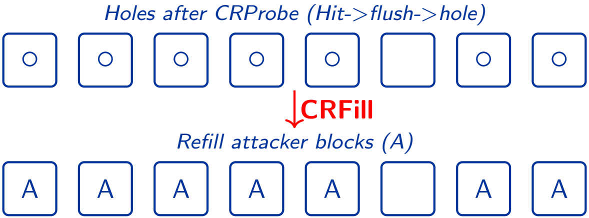
Fingerprinting using Cache Footprint

Holes after CRProbe (Hit->flush->hole)

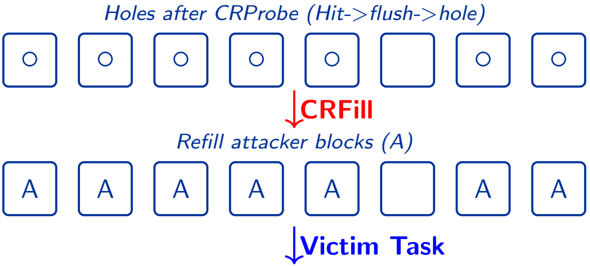
Fingerprinting using Cache Footprint



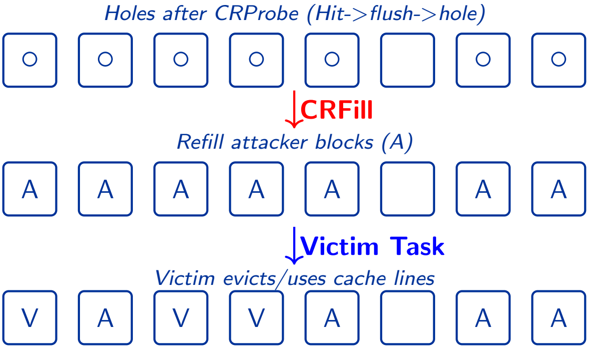
Fingerprinting using Cache Footprint



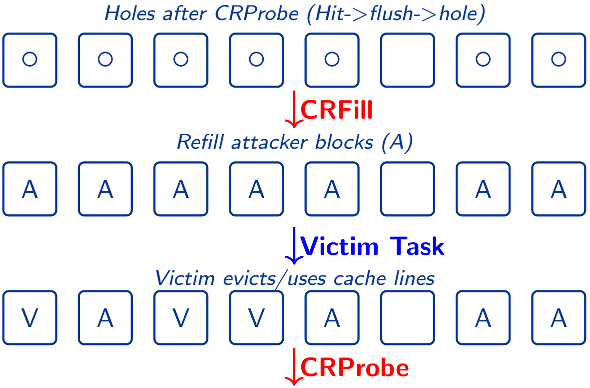
Fingerprinting using Cache Footprint



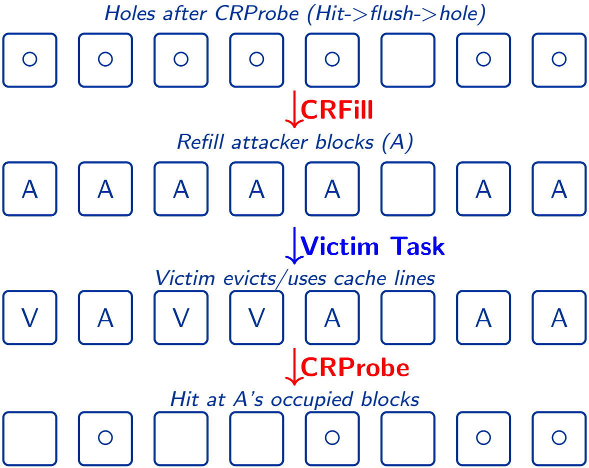
Fingerprinting using Cache Footprint



Fingerprinting using Cache Footprint



Fingerprinting using Cache Footprint



Exploring Fingerprinting in RFA-LLCs

- After validating cache fill effectiveness, we investigate the feasibility of **fingerprinting attacks** on RFA-LLC in full-system mode.

Exploring Fingerprinting in RFA-LLCs

- After validating cache fill effectiveness, we investigate the feasibility of **fingerprinting attacks** on RFA-LLC in full-system mode.
- Fingerprinting aims to infer system activity by detecting disturbances to attacker-controlled cache blocks.

Exploring Fingerprinting in RFA-LLCs

- After validating cache fill effectiveness, we investigate the feasibility of **fingerprinting attacks** on RFA-LLC in full-system mode.
- Fingerprinting aims to infer system activity by detecting disturbances to attacker-controlled cache blocks.

Key Challenge in Realistic Setups

Realistic setups introduce **non-deterministic noise** due to OS and background processes.

Experimental Procedure: OS Noise Analysis

- Attacker executes **CRFill** and **CRProbe** to occupy a significant portion of the LLC.
- Then, the process goes inactive for a controlled interval (x ms), allowing only OS/background activities.

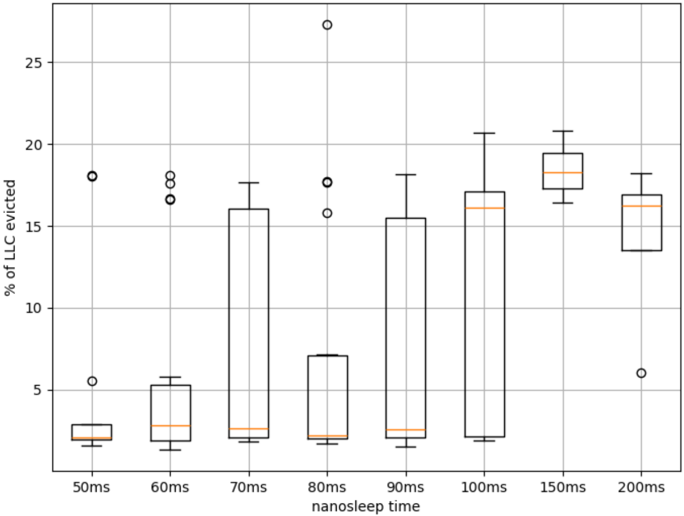
Experimental Procedure: OS Noise Analysis

- Attacker executes **CRFill** and **CRProbe** to occupy a significant portion of the LLC.
- Then, the process goes inactive for a controlled interval (x ms), allowing only OS/background activities.
- We conduct this using **two methods of inactivity**:
  - `nanosleep()` system call
  - `rdtsc`-based busy wait using a `while` loop

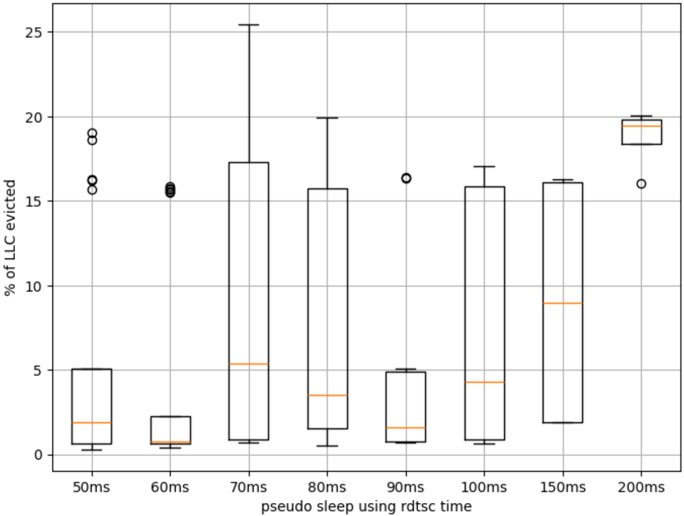
Experimental Procedure: OS Noise Analysis

- Attacker executes **CRFill** and **CRProbe** to occupy a significant portion of the LLC.
- Then, the process goes inactive for a controlled interval (x ms), allowing only OS/background activities.
- We conduct this using **two methods of inactivity**:
  - `nanosleep()` system call
  - `rdtsc`-based busy wait using a `while` loop
- After x ms, attacker **probes** the cache again to estimate how many of its blocks were disturbed.
- This helps infer the **extent of OS-induced eviction** over time.

Footprint due to sleep



Footprint due to pseudo sleep



Experimental Procedure: OS Noise Analysis

Observation

A longer idle interval $\rightarrow$ more blocks disturbed $\rightarrow$ higher system activity footprint.

However, **the presence of OS noise introduces considerable variability** in the observed cache behavior, impacting the consistency of attack performance.

Conclusion

Key Takeaway

Operating system noise significantly increases variability in cache occupancy, leading to observable degradation in attack performance.

- The attacker can still achieve **substantial occupancy** across diverse access patterns even in nosiy settings.
- However, **system-level interference** introduces **non-determinism** in cache behavior, impacting repeatability and effectiveness of occupancy-based attacks.

Future Work

Next Steps

- Investigate techniques to **mitigate the impact of OS-induced noise** on cache behavior to carry out cache occupancy based attacks.
- Explore **adaptive or noise-resilient attack strategies** that can maintain effectiveness even under significant system-level interference.

Thank you for Attending!

Questions and Answers!