UGP Report
LLCWhisp: Feasibility of Occupancy-Based Attacks on Fully Associative LLCs with Random Eviction

Abhishek Khandelwal (220040)
Indian Institute of Technology Kanpur
India
abhishekkh22@iitk.ac.in

Pallav Goyal (220747)
Indian Institute of Technology Kanpur
India
pallavg22@iitk.ac.in

Abstract

Many attacks leverage the timing channel vulnerability of shared last-level cache (LLC) in multi-core systems. To mitigate conflict-based side channel attacks in inclusive and shared LLCs, recent studies have proposed to use fully associative LLCs with random evictions (RFA-LLC). Moreover, it is also established that occupancy based attacks on RFA-LLC are possible. Occupancy-based attacks, after occupying the LLC to an extent, rely heavily on observing and differentiating levels of disturbance to draw inferences which can be used to create covert communication channels and launch fingerprinting attacks. The scope of this project is to study the feasibility of such attacks in presence of Operating System (OS) scheduling and other background activities.

In this project, first, we implement and validate a fully associative LLC with random replacement policy in the gem5 full-system simulator. To study the effectiveness of cache filling procedure proposed by an earlier study from the research group, we examine the achievable occupancy of the LLC under various access patterns and cache-filling strategies with the Linux OS on gem5 full-system simulator. We observe that the cache filling procedures are minimally impacted by OS and other background activities. Next, we examine the possibility of cache finger printing attack in this setup. However, we observe significantly differing results in gem5 full system setup compared to earlier studies because of the noise introduced by OS and other background activities. To study the extent of OS intervention, we performed experiments to observe the occupancy after different intervals of inactivity in user space to isolate OS and other background activities, and find that the footprint of other execution grows significantly beyond an interval of 100 milliseconds. We conclude that occupancy attacks in RFA-LLCs in realistic setups need to address the challenges introduced due to OS and other background activities.

1 Introduction

Modern multi-core processors rely on a shared last-level cache (LLC) to improve memory bandwidth and reduce access latency. In an inclusive, shared LLC design, all cores can evict or refill any cache line, creating an implicit communication channel: subtle timing differences between cache hits and misses can be observed by an attacker to infer another core's memory access patterns. Such *timing-channel* and *occupancy-based* side-channel attacks have been demonstrated to leak sensitive information (e.g., cryptographic keys or process fingerprints) even when higher-level software protections are in place.

To defend against conflict-based side-channels, recent work has proposed *fully associative* LLCs with *random eviction* (RFA-LLC), which eliminate set-indexed replacement and randomize eviction victims. While random eviction frustrates targeted line eviction, it does not prevent an attacker from *occupying* the cache and then observing the aggregate disturbance caused by the victim's activity. In an *occupancy-based* attack, the adversary first "fills" the LLC with its own data, allows the victim to execute and evict a subset of those lines, and then "probes" the cache to count how many attacker lines remain. The resulting occupancy differential serves as a statistical channel for covert communication or fingerprinting.

Most prior evaluations of occupancy attacks on RFA-LLCs have been conducted in highly controlled or simulation-only environments, neglecting the effects of an operating system's scheduler, interrupts, and background daemons. In a realistic full-system setting, these sources of noise introduce non-determinism in cache occupancy measurements, degrading repeatability and overall attack performance.

In this work, we implement a fully associative LLC with a random replacement policy in the gem5 full-system simulator running Linux OS, and validate cache-filling and probing procedures under various access patterns. We then investigate occupancy-based fingerprinting in the presence of OS scheduling and background activity. Our results show that attackers can still achieve substantial cache occupancy—comparable to simplifies, OS noise-free simulations—across diverse access patterns. However, during the victim's execution window, OS-induced events (task scheduling, interrupts, and background processes) introduce unpredictable variation in the number of evicted cache lines, degrading the repeatability and effectiveness of occupancy-based attacks. To mitigate this, we propose minimizing the attack's execution time—thereby reducing the window for OS interference—which improves consistency and overall success in realistic environments.

2 Background and Related Work

Modern processors employ a multi-level cache hierarchy to bridge the speed gap between CPU cores and main memory. The last-level cache (LLC), typically shared among all cores, is most susceptible to side-channel exploitation due to its inclusive or shared design and relatively large size. Traditional LLCs are organized as N-way set-associative structures, using replacement policies such as least-recently-used (LRU) or pseudo-LRU to manage evictions. However, set-indexed replacement enables an adversary to target specific cache sets and evict victim lines in a controlled manner, giving rise to conflict-based timing channels

To thwart conflict-based attacks, fully associative caches with random eviction (RFA-LLC) have been proposed. In an RFA-LLC, any cache line may be evicted on a miss, chosen uniformly at random from all resident lines. This design eliminates the predictability of set indexing and disrupts fine-grained eviction strategies. [1] Nevertheless, occupancy-based side-channels remain possible: an attacker first "fills" the LLC with its own lines, allows the victim to execute and displace some of them, and then "probes" the occupancy by measuring hit/miss latencies across the attacker-owned addresses. The resulting occupancy differential can be used for covert communication or fingerprinting

Prior studies of occupancy attacks on RFA-LLCs have largely been confined to simplified environments— without an operating system or background workload. These works demonstrate the theoretical capacity of occupancy channels but do not account for the non-deterministic noise introduced by real OS schedulers, interrupts, and concurrent processes.

In summary, while the architecture and eviction strategies of fully associative random caches are well understood and occupancy-based attacks have been demonstrated in noise-free settings, there remains a gap in understanding their feasibility in full-system scenarios with realistic OS noise. This work addresses that gap by implementing an RFA-LLC in gem5 full-system mode and quantifying the impact of OS scheduling and background activity on attack reliability.

3 Implementation and Experimental Setup

We implement and evaluate our attack framework using the gem5 full-system simulator configured with the x86 instruction set architecture and the Linux operating system. The experiments are conducted in a realistic setting, where operating system scheduling, context switches, and background activity introduce natural system noise. This allows us to assess the feasibility of occupancy attacks under practical constraints.

3.1 Gem5 Configuration

Table 1 summarizes the gem5 system configuration used in our experiments. The memory hierarchy includes two private L1 caches (instruction and data), a private L2 cache, and a shared last-level cache (LLC). The LLC is configured to be fully associative and utilizes a custom random replacement policy. To ensure that cache occupancy behavior is not influenced by speculative memory accesses, all hardware prefetchers were explicitly disabled.

Table 1: Gem5 System Configuration

Parameter	Value
ISA	x86
CPU Frequency	3 GHz
Cores	2
L1-I, L1-D	32KB, 8-way, 64B line, 1 cycle, LRU
L2	256KB, 16-way, 64B line, 3 cycles, LRU
LLC (L3)	2MB, 32768-way, 64B line, 15 cycles, RandomRP
DRAM	SingleChannel DDR3 1600, 3GB

3.2 Fully-Associative LLC with Random Replacement

To simulate a fully-associative last-level cache, we configure the LLC as a single-set cache with associativity equal to the total number of cache blocks:

$$\text{Associativity} = \frac{\text{Cache Size}}{\text{Block Size}} = \frac{2\text{MB}}{64\text{B}} = 32768$$

A key challenge in this setup is implementing an efficient random replacement policy. To enable constant-time selection of random blocks, we extend the gem5 cache implementation to maintain an auxiliary data structure—**the Invalid Blocks List**. This list tracks all invalidated cache blocks and allows constant time insertion and removal as blocks are invalidated or reused. On a cache miss, the replacement logic follows the flow illustrated in Figure 1.

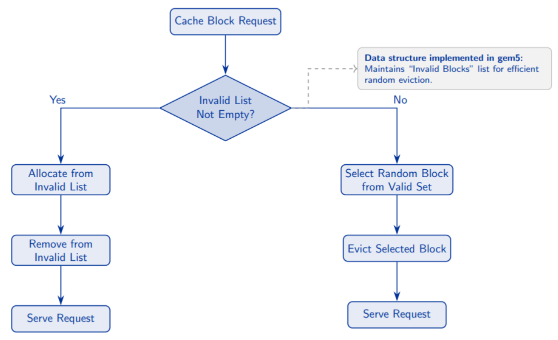
Cache Block Request
Data structure implemented in gem5: Maintains "Invalid Blocks" list for efficient random eviction.
Yes
Invalid List Not Empty?
No
Allocate from Invalid List
Select Random Block from Valid Set
Remove from Invalid List
Evict Selected Block
Serve Request
Serve Request

Figure 1: Eviction logic in FA-LLC with random replacement policy

When a cache block request occurs:

- If the invalid list is non-empty, a block is allocated from this list and removed upon use.
- If no invalid block is available, a random block from the valid set is selected and evicted to make space.

This structure ensures uniform eviction probabilities while preserving efficiency, enabling us to realistically simulate a random eviction LLC.

3.3 System Noise and OS Interference

All experiments are executed under full-system mode with Linux OS booted and running background processes. This setup introduces realistic noise from context switches, scheduler decisions, kernel daemons, and interrupt handling. By maintaining this natural interference, we aim to capture the challenges faced by an attacker in a real-world environment—particularly the variability introduced during the victim's execution window, which affects eviction-based occupancy measurements.

4 Experimental Methodology and Results

We evaluate the feasibility of occupancy-based attacks on a fully associative LLC with random replacement under realistic system conditions using a `gem5` full-system simulation environment running a Linux OS. Our methodology focuses on validating cache occupancy manipulation and probing under OS-induced noise and background activity.

4.1 Measuring Cache Occupancy and Probing

We define two core primitives: **CRFill** and **CRProbe** (illustrated in Figure 2) to mount the attack.

- **CRFill** aims to occupy a large fraction of the LLC by repeatedly accessing attacker-controlled data. This creates a high cache occupancy baseline.
- **CRProbe** then selectively accesses the same data to measure hit/miss latencies, which allows us to estimate how many cache lines remain under attacker control (i.e., the occupancy).

To determine whether a memory access results in a cache hit or miss, we rely on access timing thresholds using the `rdtsc` instruction. Pseudo code of the probing algorithm is shown below:

Algorithm 1 Cache Occupancy Probing (CRProbe)

```
1:  $i \leftarrow$ arr_size $- 1$
2:  **while** $i \geq 0$ **do**
3:      $t_1 \leftarrow$ rdtsc()
4:      load($arr[i]$)
5:      delay $\leftarrow$ rdtsc() $- t_1$
6:      clflush($arr[i]$)
7:      **if** delay $<$ threshold **or** $i =$ arr_size $- 1$ **then**
8:          occ $\leftarrow$ occ $+ 1$
9:      **end if**
10:     $i \leftarrow i - 1$
11: **end while**
```

Misses are carefully handled using `clflush` to avoid unintentionally evicting lines that are still occupied—a crucial consideration in systems with random replacement policies.

4.1.1 Experimental Design

To assess cache fill effectiveness under OS noise:

- Multiple access patterns were tested, denoted as x^a — where x represents the size of the accessed data (as a multiple of the LLC size), and a denotes the number of complete iterations over the data.

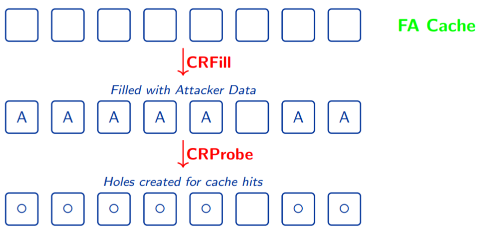

Figure 2: Illustration of CRFill and CRProbe primitives for measuring cache occupancy

All evaluations are run with the Linux OS fully operational, without disabling any kernel daemons or services. Importantly, **the prefetcher was turned off** in our `gem5` configuration to avoid artificially boosting cache occupancy.

Our primary metric is **achievable occupancy**, which reflects the percentage of cache lines that remain filled with attacker-controlled data after the CRFill operation.

4.1.2 Results and Observations

The occupancy trends under varying fill strategies are shown in Figures 3, 4, and 5. Results highlight:

- A sharp rise in occupancy with increasing data size and access repetitions, confirming the effectiveness of static CRFill patterns.
- The $(1.2x)^9$ access pattern emerges as near-optimal, achieving over 99% occupancy while avoiding significant over-access.
- Dynamic CRFill — with access sequence of $(1x)(1.1x)^2(1.2x)^6$ — also reaches over 99% occupancy, despite using fewer total accesses, demonstrating efficiency in achieving saturation.

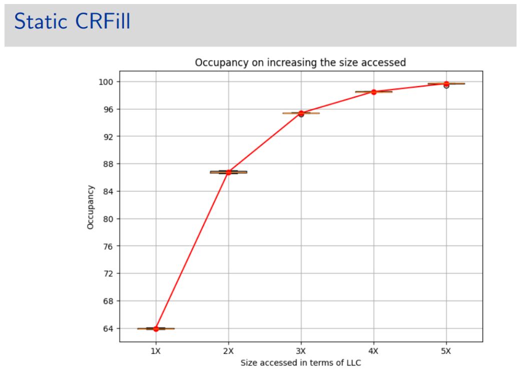

Figure 3: Occupancy vs. accessed data size (x) under static CRFill

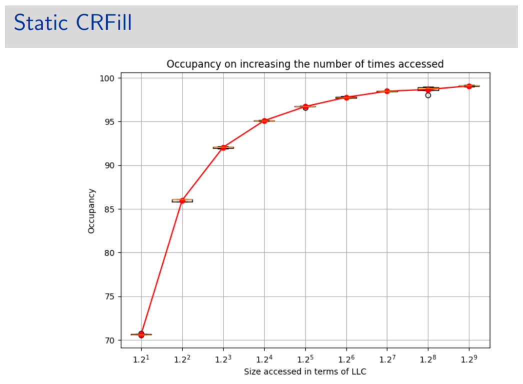

Figure 4: Occupancy vs. access repetitions (a) under static CRFill (1.2^a patterns)

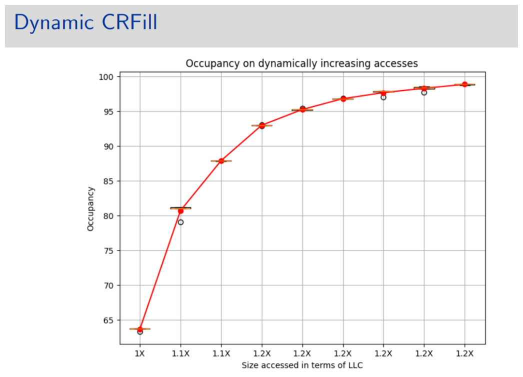

Figure 5: Occupancy under dynamic CRFill: $(1\mathbf{x})(1.1\mathbf{x})^2(1.2\mathbf{x})^6$ access sequence

Overall, attacker is able to achieve **over 99% cache occupancy** using carefully chosen access patterns, even in the presence of OS-induced noise. This validates the feasibility of occupancy-based attacks in realistic, noisy environments without requiring perfect isolation.

4.2 Fingerprinting using Cache Footprint

After validating cache occupancy effectiveness, we investigate the feasibility of fingerprinting attacks in RFA-LLC under full-system simulation with realistic OS noise. Fingerprinting refers to inferring system activity by observing disturbances in attacker-controlled cache blocks. In realistic environments, such activity detection becomes challenging due to non-deterministic interference from OS and background processes.

4.2.1 Experimental Setup

We evaluate fingerprinting by leveraging cache occupancy patterns using `CRFill` and `CRProbe`, as illustrated in Figure 6. The attacker first performs an initial `CRFill` followed by a `CRProbe` to determine the baseline occupancy—i.e., how much of the LLC is successfully filled with attacker-controlled data, as shown in Figure 2. This provides a reference point for later comparison.

To ensure the fingerprinting test begins with maximum attacker control, a second `CRFill` is executed to fully occupy the cache before entering the idle interval. The entire setup phase—initial fill, probe, and final fill—takes approximately **1.3 milliseconds**, keeping the attacker activity short and consistent across all trials. This minimizes the window for unintended OS disturbance before entering the fingerprinting phase.

Subsequently, the attacker goes inactive for a controlled interval x, allowing only OS and background processes to execute and potentially disturb the cache. Two forms of inactivity are tested:

- **`nanosleep()`** system call, which yields execution to the scheduler.
- **Busy-waiting** using `rdtsc`-based time loops (pseudo sleep), where the process remains active but performs no memory operations.

After the sleep interval, the attacker re-executes `CRProbe` to detect which of its blocks remain in the cache. By comparing this with the initial occupancy, the attacker can estimate the extent of eviction caused by system activity, thereby inferring the level of background interference during the sleep period.

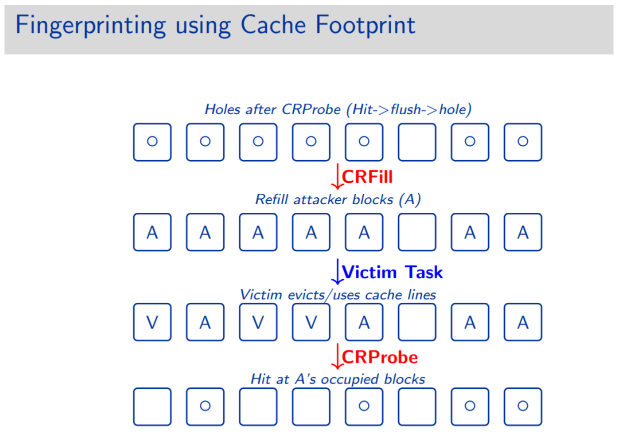
Fingerprinting using Cache Footprint
Holes after CRProbe (Hit->flush->hole)
CRFill
Refill attacker blocks (A)
A
Victim Task
Victim evicts/uses cache lines
V
CRProbe
Hit at A's occupied blocks

Figure 6: Fingerprinting using cache footprint: attacker detects victim activity by observing disturbances in its occupied cache blocks.

4.2.2 Results and Observations

Figures 7 and 8 show the percentage of attacker-controlled cache lines evicted during various sleep intervals using `nanosleep()` and pseudo sleep respectively. We observe:

- A longer idle interval results in more cache lines being disturbed, indicating increased system activity footprint.
- However, the extent of eviction varies significantly across trials due to OS noise, impacting the consistency and reliability of fingerprinting results.

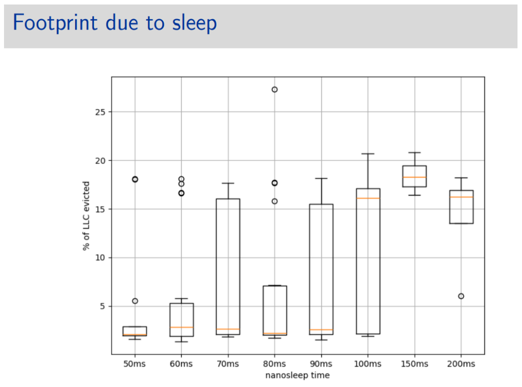

Figure 7: LLC footprint due to `nanosleep()`: percentage of attacker lines evicted across different idle durations.

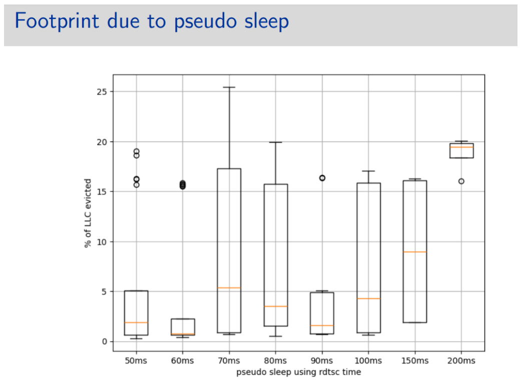

Figure 8: LLC footprint due to `rdtsc`-based pseudo sleep: higher variance observed due to busy-waiting.

4.3 Cache Fingerprinting via MD5 File Hashing

To further investigate the feasibility of fingerprinting using cache occupancy patterns, we perform experiments where a victim process computes MD5 hashes on files of varying sizes. The attacker, after performing an initial `CRFill` and `CRProbe`, attempts to infer the size of the file accessed by the victim based on observed evictions in the LLC.

4.3.1 *Experimental Setup:*

- The attacker occupies a large portion of the LLC and measures baseline occupancy using the `CRProbe` algorithm.
- A victim process then computes an MD5 hash over a file of a specific size.
- Once the MD5 hashing completes, the attacker probes the cache again to determine how many of its cache lines have been evicted.
- This process is repeated for file sizes ranging from 8KB to 256KB, and the resulting LLC evictions are recorded.

4.3.2 ***Goal:*** The aim is to establish whether the attacker can distinguish the size of a file hashed by the victim, using the percentage of LLC lines evicted as a side-channel signal.

4.3.3 ***Implementation:*** The experiment uses a custom C program that forks into attacker and victim processes. The attacker uses time-stamped memory access measurements (`rdtsc`) to identify evictions and communicate via pipes to synchronize phases. The victim uses OpenSSL's `MD5_Update` function on memory-mapped files of varying sizes.

4.3.4 ***Results:*** Figure 9 shows the percentage of LLC blocks evicted for each file size. Larger files tend to evict more cache blocks, producing distinguishable eviction patterns. However, variability remains due to system noise.

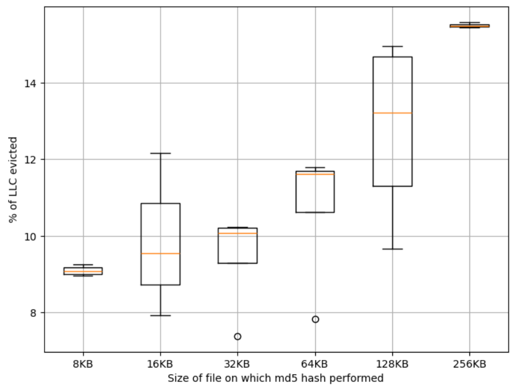

Figure 9: Eviction patterns (% of LLC evicted) during MD5 hashing of files of varying sizes

4.3.5 ***Inference:*** While the general trend shows higher eviction for larger file sizes, we observe significant overlap in the eviction distributions for `8KB` to `128KB`. This highlights the challenge of distinguishing small workloads under OS noise. However, larger file sizes (e.g. `256KB`) produce more distinguishable eviction patterns. This suggests that:

- OS-induced variation remains a key challenge for reliable fingerprinting.
- As the footprint of the victim increases, the resulting cache disturbance becomes more distinct, enabling classification of higher-load tasks.
- Fine-grained distinction between smaller file sizes is difficult due to overlapping eviction patterns caused by system-level interference.

5 Conclusion

This work investigates the feasibility of occupancy-based fingerprinting attacks on fully-associative last-level caches with random eviction policy (RFA-LLCs) under realistic, full-system conditions. Our implementation within the gem5 full-system simulator reveals that while attackers can achieve high cache occupancy through carefully designed access patterns, fingerprinting attacks face significant challenges in practice.

A central challenge is the unpredictable interference introduced by operating system activities such as scheduling, interrupts, and background daemons. Our controlled experiments—where the attacker goes inactive using either `nanosleep()` or `rdtsc`-based busy-waiting—demonstrate that even modest idle intervals (beyond 100 milliseconds) allow substantial portions of the cache to be disturbed. This variation makes it difficult for the attacker to reliably infer system activity based on cache footprint, thereby limiting the repeatability and effectiveness of such attacks in real-world settings.

To further understand the attack feasibility, we also explored a practical fingerprinting scenario involving the execution of MD5 hashing tasks on files of varying sizes. The attacker, after filling and probing the cache, observes the eviction patterns induced by the victim's MD5 computation. Our results show that while larger file sizes (e.g., 256KB) lead to consistently higher eviction percentages, overlaps in cache footprint for smaller file sizes (8–64KB) hinder accurate classification. This underscores that although coarse-grained fingerprinting is achievable, fine-grained inference remains unreliable due to the inherent variation introduced by OS noise and system-level interference.

Despite using efficient eviction-agnostic probing techniques such as `CRFill` and `CRProbe`, the observed cache behavior remains highly variable across repeated runs. This variability confirms that occupancy-based attacks in RFA-LLCs are severely impacted by OS-level noise.

To improve consistency, we propose that such attacks should minimize their execution time—thereby reducing the exposure window to OS interference. However, even this is not sufficient to eliminate the inherent non-determinism introduced by realistic system behavior.

In summary, our findings reveal that while randomized eviction in RFA-LLCs complicates traditional side-channel and fingerprinting attacks, it does not eliminate them entirely. System noise remains a dominant factor in attack viability, and effective exploitation demands both precise timing and adaptive strategies to reduce the influence of external disturbances.

6 Future Work

Our study reveals the significant role of operating system noise in degrading the effectiveness of occupancy-based fingerprinting attacks on RFA-LLCs. While we highlight the challenges posed by real-world interference, several promising avenues remain for further exploration.

First, future work can focus on mitigating the effects of OS-induced noise by leveraging system-level mechanisms that reduce interference during attack execution. Techniques such as CPU core isolation (e.g., using `isolcpus` or `cpuset` in Linux), redirecting interrupts away from the attacker's core, and assigning real-time scheduling policies can help create a relatively quiet execution environment. Additionally, attackers may monitor system activity to opportunistically launch fingerprinting during naturally low-interference periods. Such OS-aware adaptations can significantly improve the consistency of cache occupancy measurements without requiring unrealistic assumptions about system quiescence.

Second, the design of adaptive or noise-resilient attack strategies represents an important direction. Rather than relying on fixed patterns or assumptions of deterministic cache behavior, such techniques could dynamically calibrate thresholds, adjust probing granularity, or use statistical filtering to better tolerate variability. These approaches may enable more reliable inference even in the presence of frequent evictions caused by background processes.

Overall, while this study emphasizes the limits of current occupancy-based fingerprinting in full-system setups, it also motivates the need for more sophisticated, timing-aware, and interference-tolerant attack designs going forward.

Acknowledgments

We extend our heartfelt gratitude to *Prof. Debadatta* and *Prof. Mainak* for their invaluable guidance and unwavering support throughout this project, especially for their commitment to regular meetings despite their busy schedules.

We also wish to express our sincere thanks to our mentor, *Ms.Yashika*, for her in-depth knowledge and continuous assistance, which greatly enriched our understanding of the subject matter. LLC Whisper is inspired from her research work ***LeakyRand***. Additionally, we would like to acknowledge *Mr. Arun* for helping us in getting familiar to gem5.

References

[1] 2023. *ASIA CCS '23: Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security* (Melbourne, VIC, Australia). Association for Computing Machinery, New York, NY, USA.