Exploiting TLBs in Virtualized GPUs for Cross-VM Side-Channel Attacks

Paper by:
Hongyue Jin, Clemson University
Yanan Guo, University of Rochester
Zhenkai Zhang, Clemson University

Presentation by:
Athena Katsanou

Published to:
Network and Distributed System Security (NDSS) Symposium ‘26
23- 27 February 2026 , San Diego, CA, USA

Contents

Motivation

- Cloud providers, like NVIDIA, sell partitions of powerful GPUs among **multiple users** on VMs.
- The user is given the impression of an isolated environment / Virtual box.

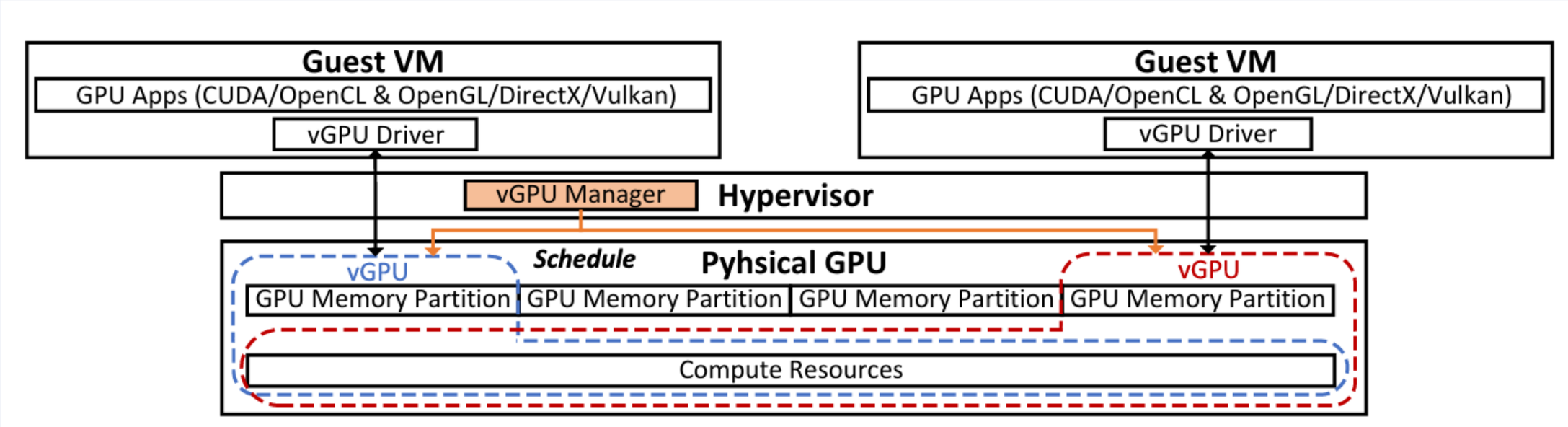
But what's the truth?

Motivation

The truth: Some resources are shared between users.

Why? The architecture of the GPU and the hierarchy of the TLBs.

The problem: The environment is not fully isolated, so one user may have access to information of another user.



In this paper

- The first side-channel attacks in virtualized GPUs in cloud settings.
- First to take advantage of the TLB architecture.
- Two cross-VM side-channel attacks in cloud settings:
  - A cheating exploit in the game Counter-Strike 2 that reveals hidden opponents
  - And a website fingerprinting attack that identifies web pages browsed by users of virtual desktops

Challenges

Undocumented Black Boxes

- SOLUTION: Reverse-engineer the specifications of the GPU.

vGPUs enforce temporal partitioning that serializes contexts (no parallel executions)

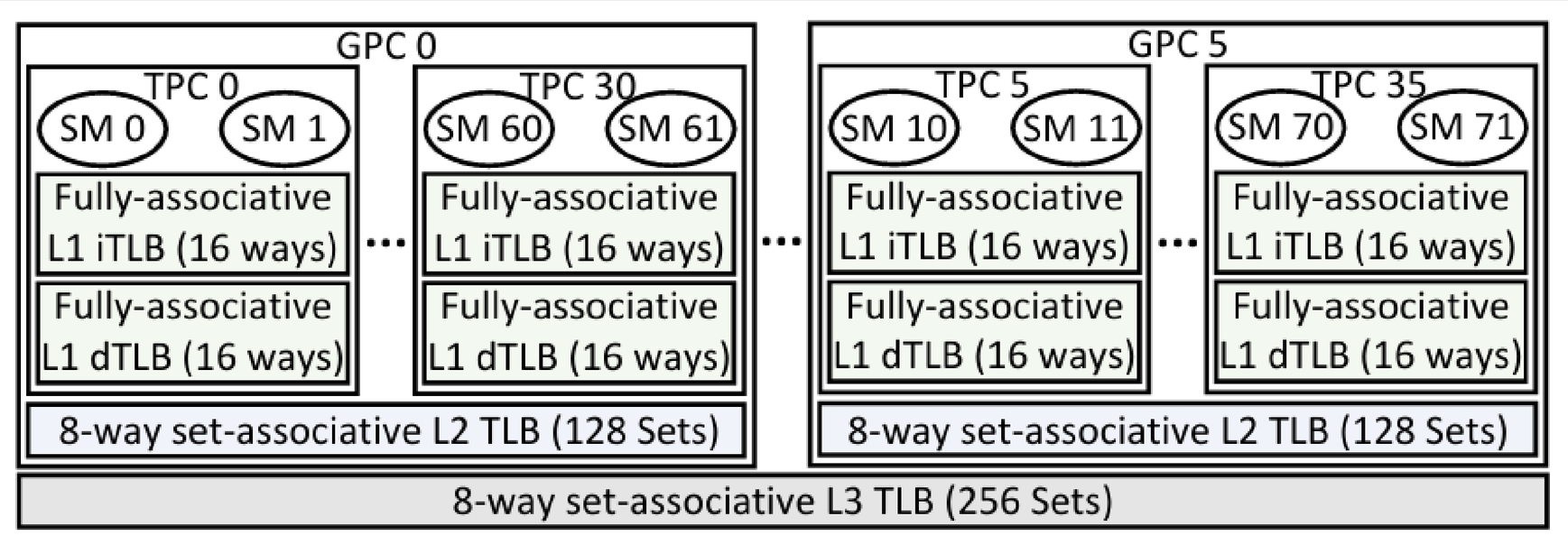
- SOLUTION: New technique with synchronizations, prime/probing mechanisms and context switch detection.

Cached PTEs (Page Table Entries) can interfere with timing measurements

- SOLUTION: Evict the GPU cache.

Background

6 GPC – Graphics Processing Clusters
36 TPC – Texture Processing Clusters
72 SM – Streaming Multiprocessors
Highly multithreaded. Execute warps – groups of 32 threads.
Follow the SIMT model (Single Instruction Multiple Threads)



QUESTIONS SO FAR?

The Threat Model

Who is the Attacker:

- They control a malicious VM that is co-resident with the victim's VM.
- They use a vGPU partition, of the same physical GPU with the victim.

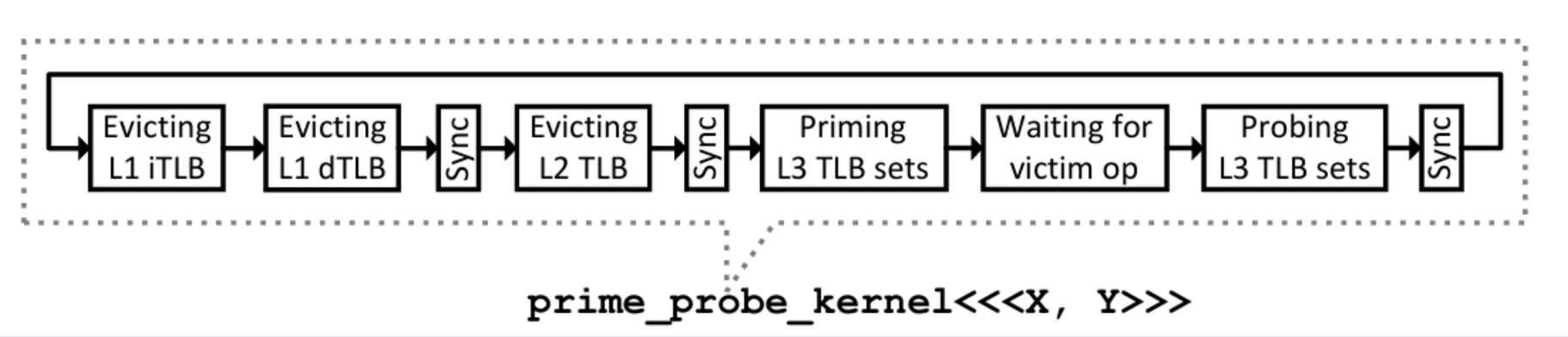
The Attacker's Goal:

- To steal sensitive data from the victim using the L3 TLB.

Capabilities & Assumptions:

- The attacker does *not* have root access to the host machine or the victim VM.
- They only have standard user access within their own VM.

The exploit in a nutshell



Eviction

Priming

Probing

Evict + Prime+Probe

Evict the L1 dTLB and L2 TLB:

- L1 dTLBs are always 16-way fully associative, so each thread in SM must access 1 of the 16 64KB pages.
- L2 TLBs have more entries (1024), so each thread needs to access 8 data pages.
- The pages are selected according to the reverse-engineered hash function

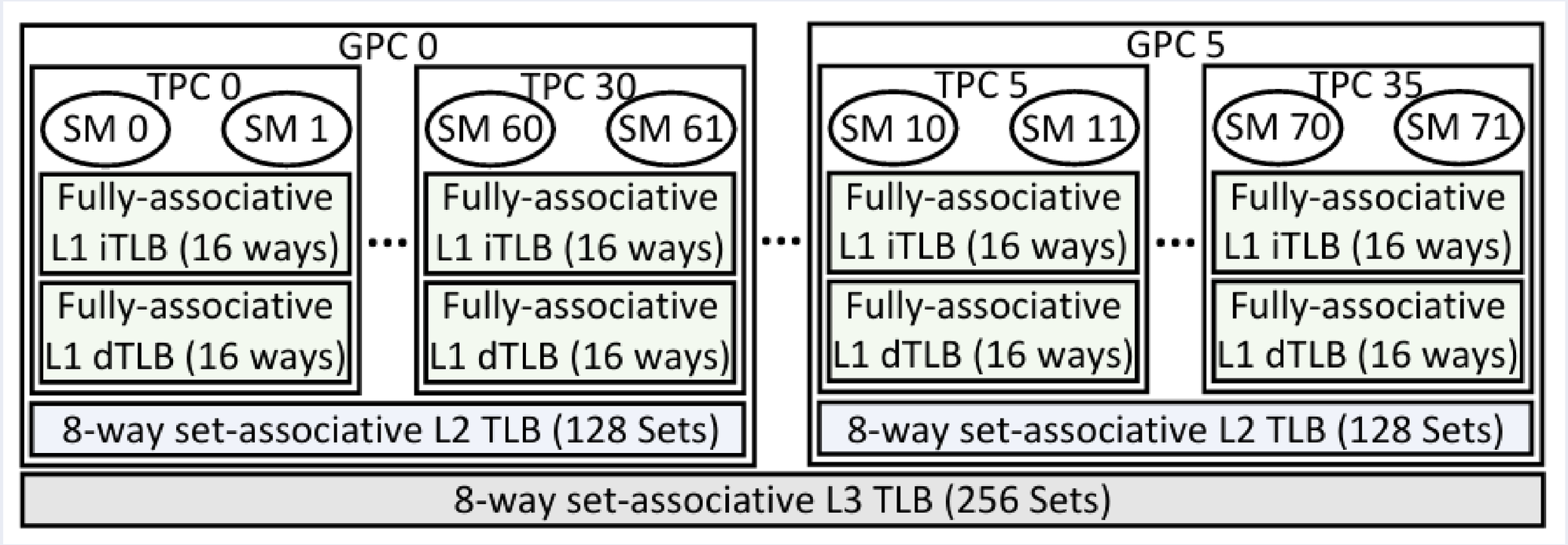
$$H_{\mathrm{L2}} = \begin{bmatrix} *****1******1******1******1******1 \\ ****1******1******1******1******1* \\ ***1******1******1******1******1** \\ **1******1******1******1******1*** \\ *1******1******1******1******1**** \\ 1******1******1******1******1***** \\ ******1******1******1******1****** \end{bmatrix}$$

Evict the L1 iTLB:

- L1 iTLBs are also 16-way fully associative, so an SM needs to access at least 16 code pages.
- CUDA uses 2MB page size for code
- They made 16 dummy functions

```
1  if (!jump_over) {
2      #pragma unroll OUTER_NUM        // e.g., 2 for Ampere GPUs
3      for (i = 0; i < OUTER_NUM; ++i) {
4          #pragma unroll INNER_NUM    // e.g., 6000 for Ampere GPUs
5          for (j = 0; j < INNER_NUM; ++j)
6              sum += clock64();
7      }
8  }
```

Architecture



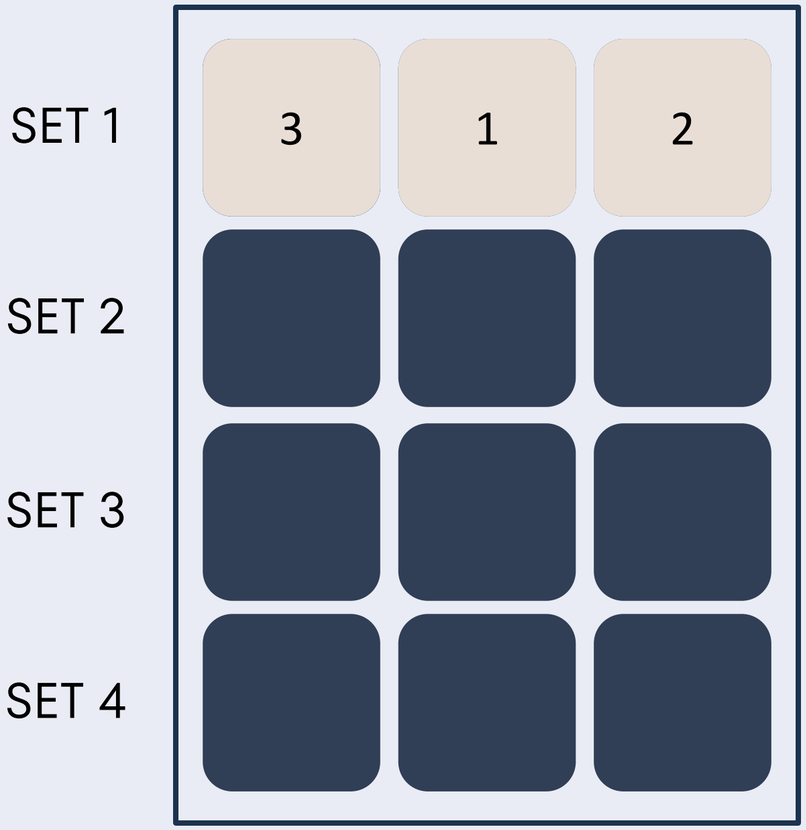
Evict Prime+Probe

- Each warp (32 threads) is responsible for priming and probing 1 TLB set.
- The warps in thread block **n** that are used to prime 1 set, mustn't be used to probe the same set.
- Probing uses the warps in thread block **(n+1) % X**.
- The assigned SMs must be in different GPCs (which means different low level TLBs).
- Each warp is serialized with CUDA's **_syncthreads()** so that they don't interfere with one another.

$$H_{\mathrm{L3}} = \begin{bmatrix} **1*******1*******1*******1 \\ *1*******1*******1*******1* \\ 1*******1*******1*******1** \\ *******1*******1*******1*** \\ ******1*******1*******1**** \\ *****1*******1*******1***** \\ ****1*******1*******1****** \\ ***1*******1*******1******* \end{bmatrix}$$

Evict

Prime+Probe



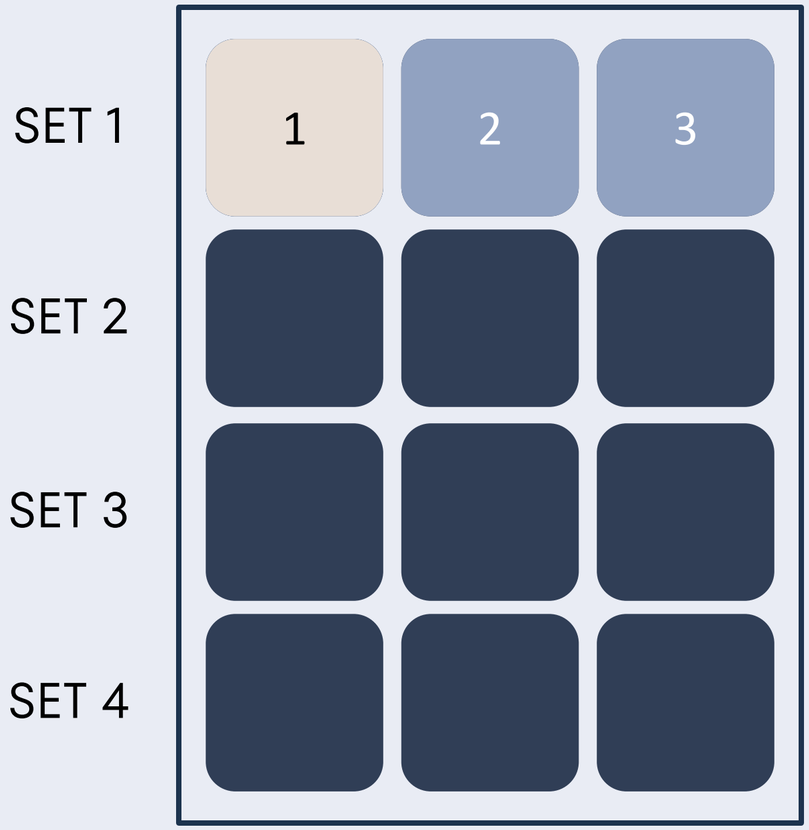
Example with 4 sets of 3 entries:

- Set 1 is primed successfully (using LRU and accessing entries 1 through 3)

- A context switch happens and the victim uses the GPU.
- They access set 1, with entry V.

- We start probing.
  - LRU logic: We access entries 1 through 3.
  - Everything gets evicted!!!

Evict + Prime+Probe



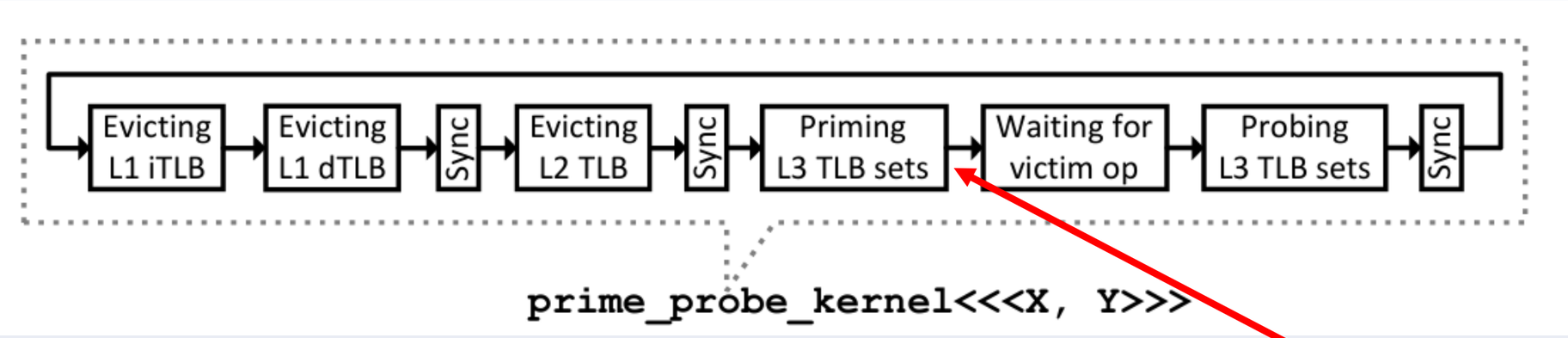
Example with 4 sets of 3 entries:

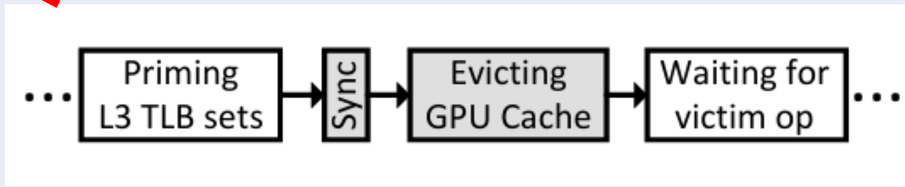
- Set 1 is primed successfully (using LRU and accessing entries 1 through 3)

- A context switch happens and the victim uses the GPU.
- They access set 1, with entry V.

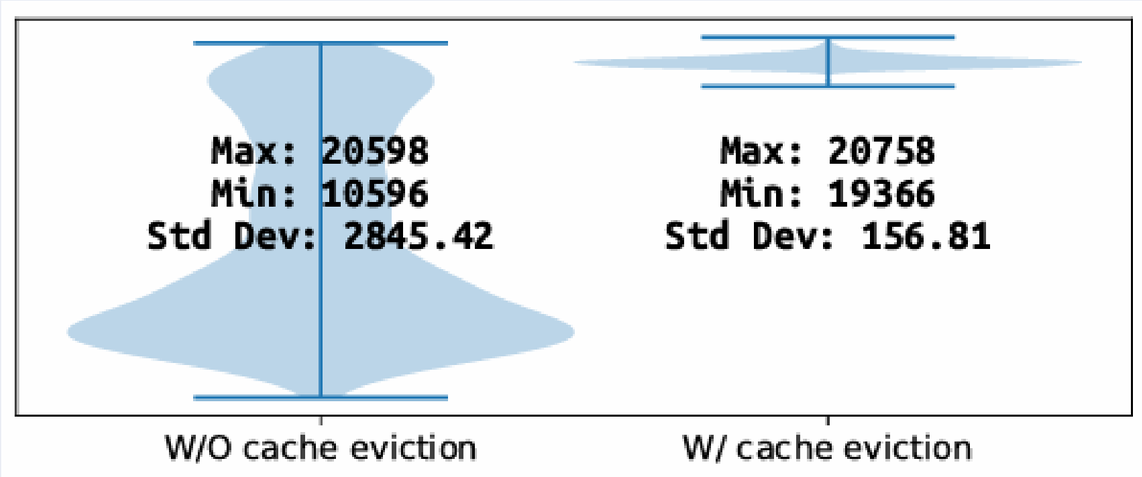
- We start probing.
  - Backwards logic: We access entries 3 through 1.
  - 3 and 2 already exist, so only 1 replaces V.
  - Success!

Enhancement



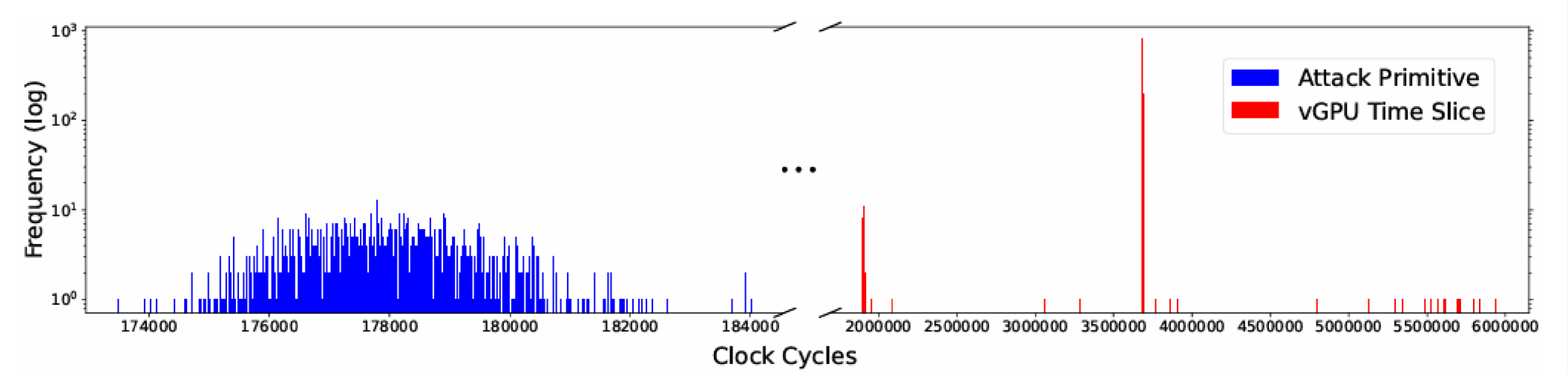


PTEs are also stored in the L2 cache, resulting in much faster page table walks.



Timings

- For the primitive to function correctly, after a context switch (attacker has access), we must successfully finish a **probing** operation and prepare for the **next round**, in 1 time slice.



- Each vGPU time slice is long enough for the attack primitive to complete its operations.
- Minimum vGPU time slice (best effort scheduling policy) = 1,700,000 cycles.

EVALUATION

Evaluation Setup

01

NVIDIA A10 GPU
For virtualization in public clouds

02

BEST EFFORT
Scheduling policy (widely used in public clouds)

03

EACH VM CONNECTED
8GB memory partition (1/3 portion of the A10 GPU)

04

ATTACKER
VM with Ubuntu 20.04

05

VICTIM
VM with Windows 11 (assumed)

06

OTHER TOOLS
CUDA and Vulkan

Website Fingerprinting Exploit

The goal:

- Identify which web pages a user is visiting in their browser on a vGPU-powered virtual desktop.

- The attack can be mounted in cloud environments without requiring direct system access (deployment of malware on the victim’s machine)

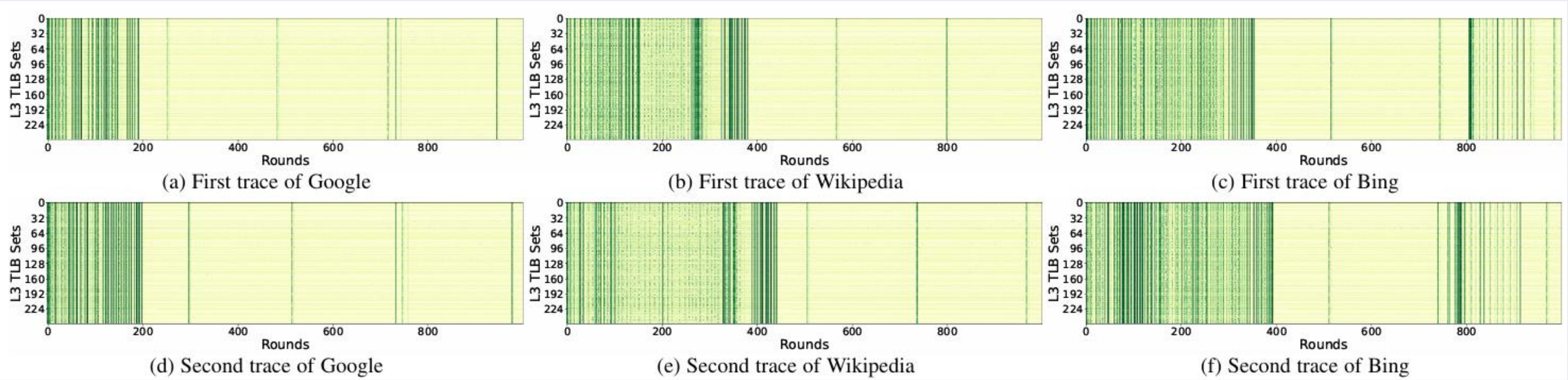
The idea:

- Different websites have different workload variations that may result in distinct patterns in GPU memory accesses.

Website Fingerprinting Exploit

- “Victim” uses the Edge browser within a Windows 11 VM
- “Attacker” collects 2 traces of the L3 TLB (1,000 Prime+Probe measurements per trace).

- Different websites with some functional or aesthetic similarities have distinctive patterns:



(a) First trace of Google (b) First trace of Wikipedia (c) First trace of Bing

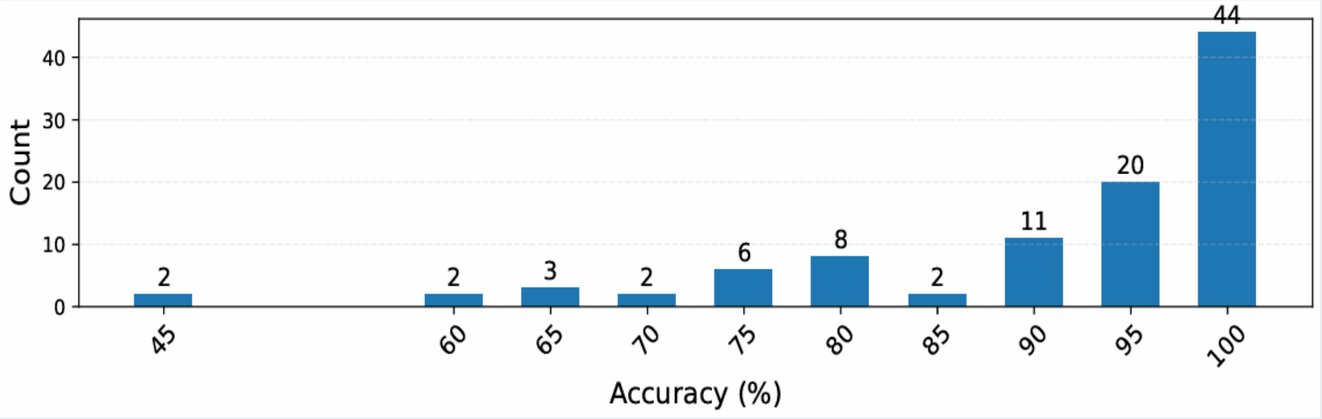
(d) Second trace of Google (e) Second trace of Wikipedia (f) Second trace of Bing

Website Fingerprinting Results

- Profiling 100 different websites
- 5,000 traces in total

} Dataset

- The victim accesses 20 times each website



	Leakage Source	Virtualized GPU	Cross-VM	#Websites	Accuracy
[25]	GPU memory allocation	✗	✗	200	90.0%
[24]	GPU memory residue	✗	✗	100	95.4%
[43]	Rendering contention	✗	✗	100	>70%
[44]	PCIe contention	✗	✗	100	96.3%
[26]	PCIe contention	✗	✗	100	95.2%
[16]	GPU L2 cache	✗	✗	50	>98%
[45]	DVFS-induced EM	✗	✗	50	85.3%
[15]	NVDEC utilization	✗	✗	40	89.2%
Ours	GPU TLB	✓	✓	100	91%

Miscalculations:

- Sites from the same company
- Social media platforms

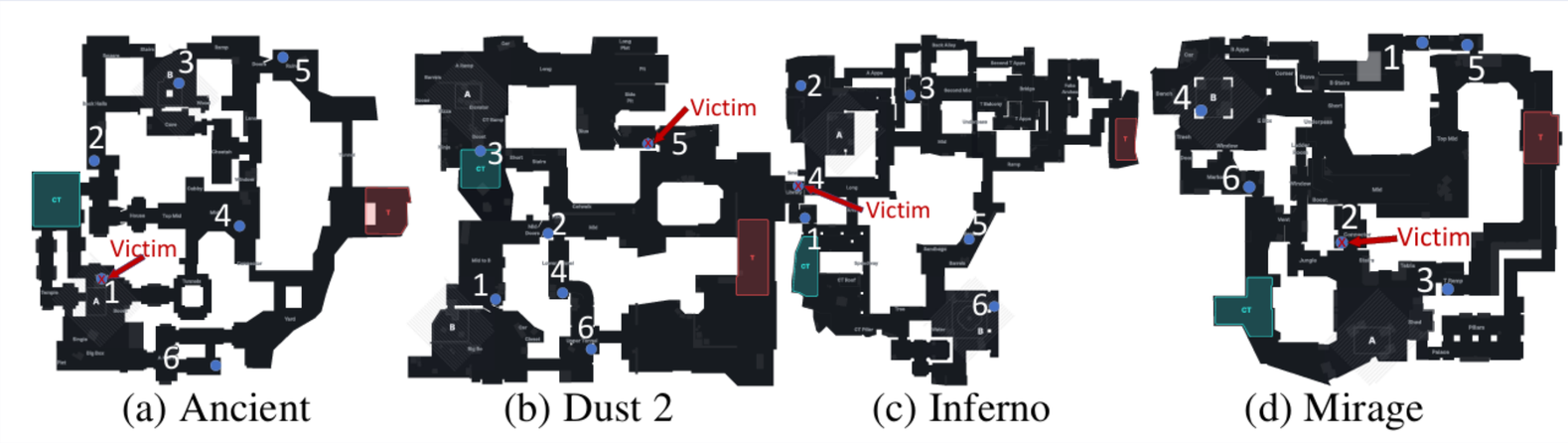
CS2 Exploit

VICTIM

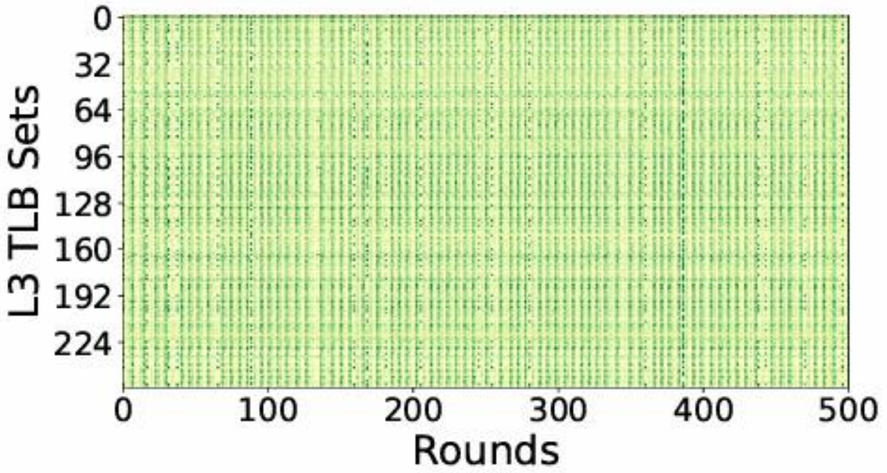
- CS player who uses a vGPU-powered VM cloud gaming service to play
- “Camping” in the game, waiting to ambush an opponent.
- Usually behind an obstacle (walls, vehicles, crates).

ATTACKER

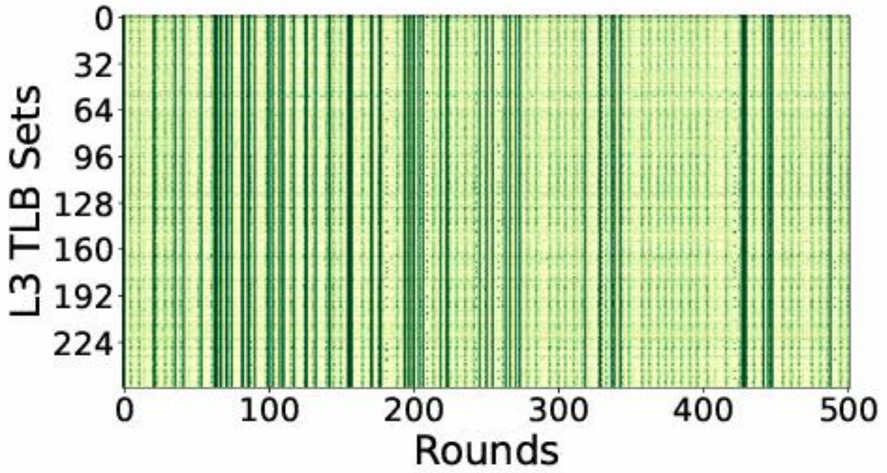
- CS player, opponent of the victim
- Has also rented a VM in the same cloud, sharing the same physical GPU as the victim’s VM (not for playing the game)
- Tries to find the victim that is hiding.



CS2 Evaluation

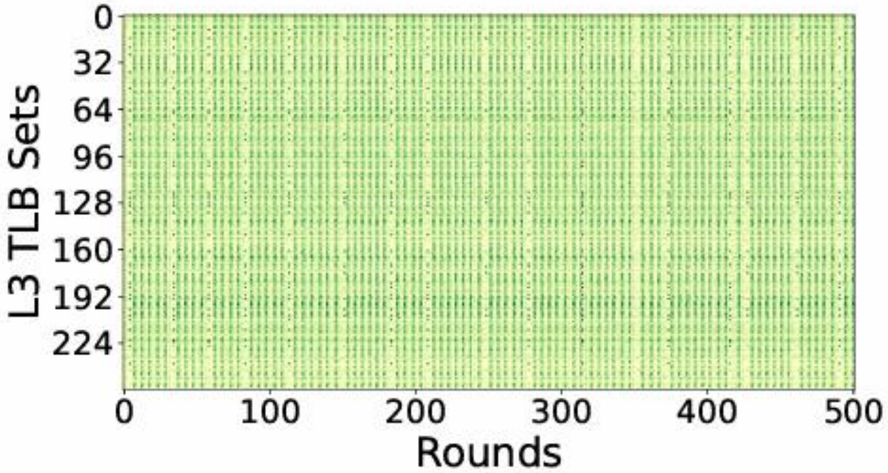


(a) Avatar stays still.

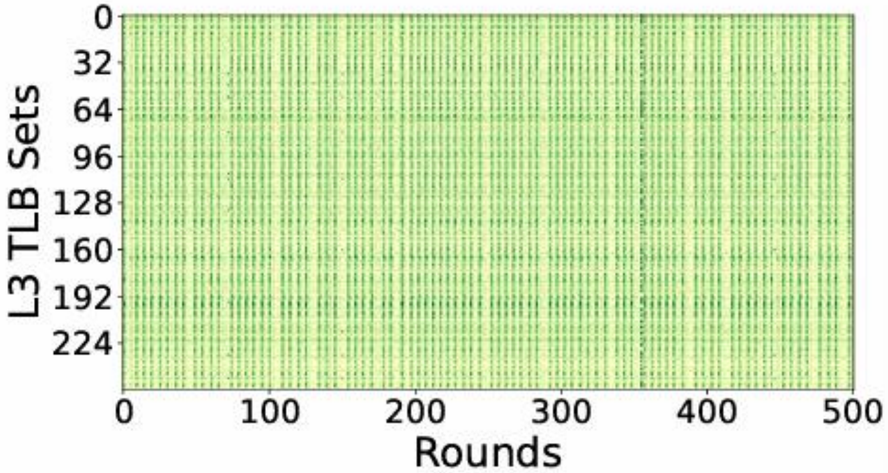


(b) Avatar moves around.

L3 TLB access patterns when a player is camping vs. moving



(a) Empty space behind the wall.

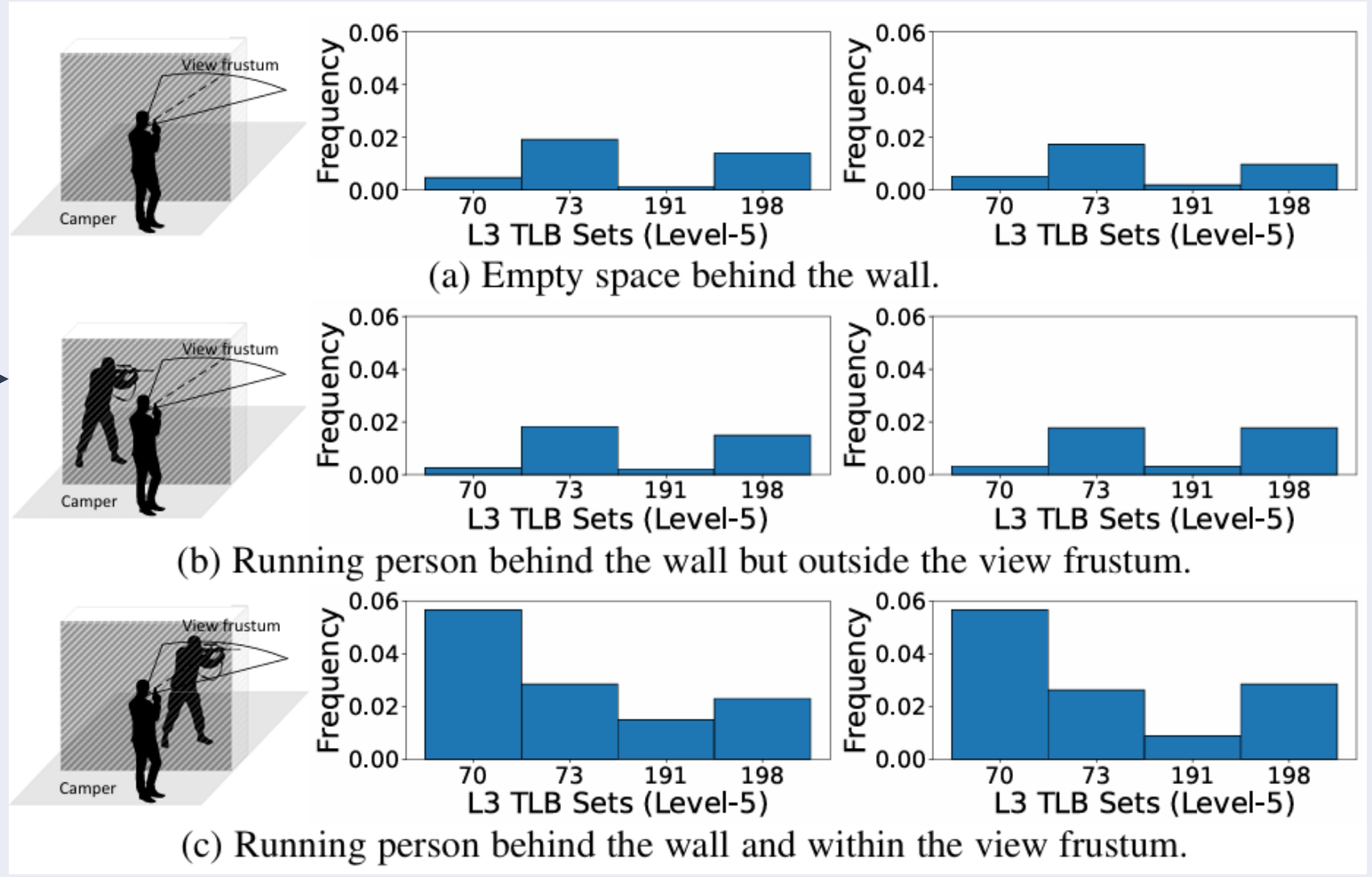


(b) Running person behind the wall.

L3 TLB access patterns when a player's view frustum has an empty space vs. an avatar running behind the wall

CS2 Evaluation

- Use of statistical distribution of the L3 TLB set contention (levels 0-to-8).
- At level-5 contention, they successfully detect an avatar behind the wall.
- The game engine still performs minimal GPU operations for occlusion testing.



Results

- Different CS maps may require different sets of L3 TLB sets for effective detection.
- They are map-specific and not system-specific (they don’t change with another Windows 11 VM).
- The contention level for the most reliable L3 TLB sets is level 4-5.
- The Euclidean distance between the frequency vector of two measurements:

TABLE II: Evaluation results for identifying the camping spot of the victim.

	Spot 1	Spot 2	Spot 3	Spot 4	Spot 5	Spot 6	Threshold
Ancient	0.14079	0.00857	0.00986	0.01286	0.01044	0.01574	≥ 0.10
Dust 2	0.01191	0.02396	0.02231	0.02005	0.12584	0.01738	≥ 0.10
Inferno	0.01454	0.02104	0.01671	0.15692	0.02433	0.00887	≥ 0.10
Mirage	0.00513	0.06224	0.00798	0.00582	0.00843	0.00674	≥ 0.03

Defences

From NVIDIA's side:

- Enhance the security of the TLB hierarchy.
- Implement a static-partition (SP) TLB or a random-fill (RF) TLB design (Hardware modification).
- Initiate a TLB shootdown at the end of each time slice. Very time consuming (Software package update).

From a tenant's side:

- Inject noise in GPU TLB access patterns to obscure the original pattern.
- Periodically monitoring access times of certain addresses. Unusually frequent L3 TLB misses may indicate an attack.

Conclusion

- The paper's idea for a **side-channel attack** was successful in the environment of VMs running on **vGPU partitions** of the same physical GPU.
- The exploit happened by using the **L3 TLB** access patterns of the **shared GPU**.
- The effectiveness of this attack primitive is demonstrated through **two case studies**.
- It highlights the **security risks** introduced by GPU sharing in virtualized environments.

QUESTIONS?

THANK YOU!