DEVIOUS: Device-Driven Side-Channel Attacks on the IOMMU

Taehun Kim*, Hyeongjin Park*, Seokmin Lee*, Seunghee Shin†, Junbeom Hur*, and Youngjoo Shin*
*Korea University †The State University of New York at Binghamton
*{taehunk, be2overflow, leeseokmin, jbhur, syoungjoo}@korea.ac.kr †sshin@binghamton.edu

2023 IEEE Symposium on Security and Privacy (SP) | 978-1-6654-9336-9/23/$31.00 ©2023 IEEE | DOI: 10.1109/SP46215.2023.10179283

***Abstract*—Modern computer systems take advantage of Input/Output Memory Management Unit (IOMMU) to protect memory from DMA attacks, or to achieve strong isolation in virtualization. Despite its promising benefits, the IOMMU could be a new source of security threats. Like the MMU, the IOMMU also has Translation Lookaside Buffer (TLB) named IOTLB, an address translation cache that keeps the recent translations. Accordingly, the IOTLB can be a target of a timing side-channel attack, revealing victim's secret. In this paper, we present DEVIOUS, a novel device-driven side-channel attack exploiting the IOTLB. DEVIOUS employs DMA-capable PCIe devices, such as GPU and RDMA-enabled NIC (RNIC), to deliver the attack. Thus, our attack has no influence on CPU caches or TLB in a victim's machine. Implementing DEVIOUS is not trivial as microarchitectural internals of the IOTLB of Intel processors are hidden. We overcome this by reverse-engineering the IOTLB and disclose its hidden architectural properties. Based on this, we construct two IOTLB-based timing attack primitives using a GPU and an RNIC. Then, we demonstrate practical attacks that target co-located VMs under hardware-assisted isolation, and remote machines connected over the RDMA network. We also discuss possible mitigations against the proposed side-channel attack.**

1. Introduction

A recent computing trend is to grant more peripheral devices direct access to the host's physical memory to relieve the burden of processors. However, this results in a series of device attacks by malicious DMA-capable peripheral devices, breaching the confidentiality and integrity of the host memory in personal computers [8], [10], [11], [19], [34], [58], [62], and breaking the isolation of virtual machines (VMs) in cloud data centers [33], [75].

The Input/Output Memory Management Unit (IOMMU) tackles these security challenges. The IOMMU is a hardware unit in the CPU that performs a DMA re-mapping, a feature that translates I/O addresses in DMA requests to physical memory addresses. With the IOMMU enabled, I/O virtualized address (IOVA) spaces are allocated for DMA devices, and all the DMA requests are subject to inspection based on their permission bits in the page table entry before translating into the physical address. The IOMMU-based protection mitigates DMA attacks as unauthorized devices are not permitted to access critical areas of the host's physical memory. In virtualization, it enables the strong isolation of VMs by separating the IOVA spaces on the basis of a group (i.e., domain) of devices.

However, IOMMU opens a new attack surface, becoming a new side-channel source between the attacker-controlled devices and a victim's device. Like MMU, the IOMMU has an address translation cache named Input/Output Translation Lookaside Buffer (IOTLB), which keeps recent IOVA-to-PA translations to serve translation requests faster. The IOTLB is shared among multiple devices across different domains. Although this has a high potential to be used as an unintentional side-channel source [22], [38], [64], [69], [79], the research communities have still not studied it.

In this paper, we present DEVIOUS, a novel side-channel attack exploiting the IOTLB in IOMMU-enabled systems. In DEVIOUS, an attacker employs his/her own DMA-capable PCIe devices, such as GPU or NIC, which share the same IOTLB with a victim's device. The attacker initiates an IOVA-based DMA request from the device, causing contention with the victim's device on the IOTLB. The measured latency of the DMA request depends on whether the request has been served by the IOTLB or not. The timing difference reveals the information on the victim's behavior. DEVIOUS is available on any configuration of the victim's device that shares the IOTLB (IOMMU). The victim's device may be on the same machine, but isolated from the attacker as they belong to different domains (e.g., VMs in a multi-tenant cloud with support of dedicated PCIe devices), or on the remote machine connected to the attacker over the network.

We classify DEVIOUS into a *device-driven side-channel attack*, as it employs DMA-capable devices to deliver the attack. Unlike CPU-based side-channel attacks [24], [25], [31], [40], [79], where memory requests to create contention on cache-like shared resources are initiated from CPUs, DEVIOUS makes *device-initiated* memory requests instead. Hence, DEVIOUS has no influence on CPU caches or TLBs on a victim's machine. With such an inherent and unique characteristic, DEVIOUS evades all the countermeasures and well-established protection techniques against CPU-based side-channel attacks, like anomaly detection based on the behavior of a cache or TLB [7], [15], [46], [81].

Implementing DEVIOUS is not straightforward as the

DOI 10.1109/SP46215.2023.00168

IOTLB is new, and its microarchitectural details are still hidden unlike TLBs and caches which have been widely studied and reverse-engineered for decades [13], [22], [57]. To bridge the knowledge gap, we reverse-engineer the IOTLB's internals in an Intel processor. We use RDMA-enabled NIC (RNIC) for the reverse-engineering as it allows us to issue a single DMA transaction, which is necessary for elaborately dissecting the internal. We first explore the RNIC's device driver to uncover the address mapping in the IOMMU. Then, we reveal the IOTLB's architectural properties by reverse-engineering it through formulated DMA transactions.

We develop two attack primitives for DEVIOUS using two devices, GPU and RNIC, considering the device usage in modern systems. The first primitive is $\text{DEVIOUS}_{\text{GPU}}$, which probes the status of IOTLB by using a GPU. It leverages the data transfer between a host and GPU device memory to issue DMA transactions on the IOMMU. The following primitive is $\text{DEVIOUS}_{\text{RNIC}}$, that utilizes RNIC for probing the IOTLB. $\text{DEVIOUS}_{\text{RNIC}}$ initiates remote DMA transactions over an RDMA network.

We prove the security impact of DEVIOUS by demonstrating attacks under practical scenarios with these two constructed primitives. An attacker can perform a side-channel attack on systems highly secured by the IOMMU. First, we leverage $\text{DEVIOUS}_{\text{GPU}}$ to construct a cross-VM covert channel and a keystroke timing attack on the SSH protocol in the virtualized environment. We show that the attack can be performed across the isolation boundary of VMs bypassing the IOMMU-based protection. Next, we use $\text{DEVIOUS}_{\text{RNIC}}$ to extend the local side-channel attack to remote attacks, where the attacker infers a victim's access pattern on the Apache Crail and performs a website fingerprinting attack remotely over the RDMA network. In the end, we discuss possible countermeasures to prevent side-channel attacks based on the computing environments.

Comparison to previous works. Unlike previous TLB-based attacks [22], [37], DEVIOUS is a device-driven attack that uses DMA-capable I/O devices to carry out the attack. Although there are similar attacks to ours, they differ in the hardware component they exploit. For instance, [38], [65] use RNIC to target the vulnerability in DDIO-enabled LLC, while [69] also uses RNIC but targets its SRAM-based internal cache. Another attack [64] exploits traffic congestion in a PCIe switch as a side-channel. In contrast, our attack leverages the IOTLB in the processor's root complex as a source of leakage. We provide a comprehensive comparison with related work in Section 8.

We underscore this paper's contributions as follows.

1) We unveil the IOMMU as a leakage source for side-channel attacks. To demonstrate it, we reverse-engineer the IOTLB's architectural properties on Intel CPUs, including cache organization and replacement policy.
2) We present a device-driven side-channel attack, DEVIOUS, exploiting the IOTLB as a side-channel, which has no influence on CPU caches or TLB, and evades countermeasures against CPU-based side-channel attacks.
3) We develop two attack primitives of DEVIOUS by using GPU and RNIC. Then, we systematically analyze and compare these primitives under experimental settings. Furthermore, we propose a method that makes the timing difference more apparent by leveraging the IOTLB's replacement policy.
4) We demonstrate practical attacks such as cross-VM and remote timing attacks on the IOMMU-enabled machines. We also discuss some effective mitigations against the IOTLB-based side-channel attacks.

2. Background

2.1. IOMMU

Intel processors with VT-d are equipped with a hardware called Input/Output Memory Management Unit (IOMMU) [30]. The purpose of the IOMMU is to protect system from DMA attacks, and achieve strong isolation of virtual machines by preventing unauthorized memory access from malicious DMA-capable devices [6], [12], [62]. Like MMU, the IOMMU allocates an I/O virtual address (IOVA) space to each DMA device. With the IOVA space, a device is isolated from the host and even other devices under different IOVA spaces. However, memory access from the device needs address translation from IOVAs to physical addresses in the host memory. DMA Remapping Hardware Unit (DRHU) inside the IOMMU is responsible for the address translation [30]. Upon an IOVA-based DMA request, the DRHU consults a page table and outputs the corresponding physical address. The page table exists in the host memory, managed by operating systems [4], [30].

The address translation is time-consuming, demanding a page-table walk with multiple memory look-ups. The DRHU looks up the page tables to find a corresponding physical address at every IOVA-based DMA request. This extra operation may hurt the system's performance with the extended DMA's latency. Like the traditional MMUs, the IOMMU thereby keeps recently accessed mappings in the Input/Output Translation Lookaside Buffer (IOTLB) to accommodate quick translation. Also, similar to the TLBs [22], the timing difference between IOTLB hit and miss can be a leakage source through a side-channel. To date, the microarchitectural internals of Intel processor's IOTLB necessary for successful side-channel attacks have not been disclosed yet.

2.2. RDMA

In modern large-scale computing, high-performance interconnected networks are commonly deployed using the Remote Direct Memory Access (RDMA) protocol to provide diverse services such as big data analysis [32], [41], deep learning [55], [66], [78], cloud service [20], [68], [77], and key-value store [14], [35], [43], [71]–[73]. RDMA is a protocol that enables direct memory access from a machine to the memory of a remote machine without involving

either machine's operating system [54]. Under the RDMA network, a client issues an RDMA request from its user space without a context switch to the kernel space, and the received server processes the request without involving the CPU. This RDMA's strategy that minimizes OS intervention in network communication is a promising solution for a low-latency network.

The RDMA protocol can be implemented with Infiniband as follows. End-to-end hosts in the network prepare a queue pair (QP) that consists of a send and a receive queue. The queues keep work requests (WR); each defines an RDMA operation, Memory Region (MR) description, and an area RNIC can access. The MR description contains all the necessary information about the device-shared memory, including a remote key. The RNIC interprets the WRs and sends an RDMA request to the remote machine. This RDMA request contains the accessed memory address and the remote key provided with the MR description. The RNIC in the remote machine authenticates the RDMA requestor by checking the key before processing the request.

2.3. Cache side-channel attack

A cache side-channel attack is a timing attack that exfiltrates confidential data (*e.g.,* secret key) using the timing disparity in the shared cache [24], [25], [31], [40], [79]. The potential side-channel source must be a shared resource and produce discernible timing disparity depending on its internal status. Generally, hardware components with internal caches for better latency can be the source [22], [38], [69].

Numerous side-channel attacks have been introduced based on how to construct and prove the status transition in the shared resource. The *flush*-based attack evicts cached data from the shared cache using a `clflush` instruction and exploits the latency difference between the hit and miss [24], [79]. The *evict*-based attack is similar to the flush-based attacks but does not rely on the `clflush` instruction when evicting the target cache line [25]. This attack evicts the target data using a cache conflict and replacement policy. The attacker prepares an eviction set, a list of addresses accessed to make the conflict with the target in the cache. In the *prime*-based attack [31], [40], the attacker primes the whole entries in the target set with the eviction set and checks that the primed blocks are intact after a while.

These cache timing attacks are commonly performed according to 3 steps. For the first step, the attacker manipulates the status of a shared resource to a specific condition for a future probe. Next, the attacker waits for a victim's behavior to the target data. Finally, the attacker probes the transition of the status by measuring timing difference.

3. Reverse-engineering IOTLB

To accomplish a successful device-driven side-channel attack on the IOTLB, we need information about the IOTLB properties, such as *cache organization* and *replacement policy*. Unfortunately, their details are still publicly unknown.

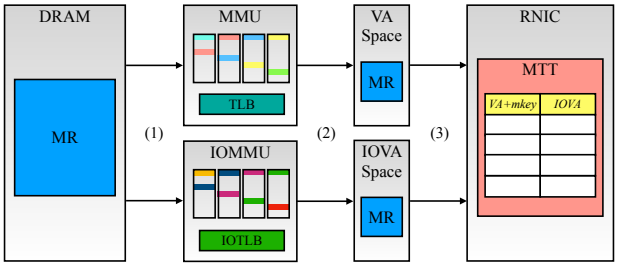


Figure 1: Procedure for updating Memory Translation Table of RNICs when IOMMU is enabled.

Thus, this work uncovers the IOTLB designs by reverse-engineering the current Intel processor. We choose Intel Xeon Silver 4316, the 3rd Generation Xeon Scalable Processor, as our target. To elaborately dissect the IOTLB, we utilize an RNIC as it allows us to generate a single DMA request at will. For this, we connected two machines with RNICs, Mellanox BlueField-2, over the network with RDMA over Converged Ethernet (RoCE) protocol. One plays a client role which sends RDMA requests to the remote machine, creating DMA transactions on the server via the RNICs. The other plays a server role, and is equipped with the target processor of reverse-engineering. We used one-sided RDMA read operations in the experiment.

3.1. Address translation mechanism in RNICs

We observed that the DMA transactions via RNICs introduce an extra challenge, requiring an additional translation from the VAs to IOVAs before issuing them. This requirement is because the transaction is requested to the RNIC with the application's VAs for the device-visible memory, but the RNIC starts DMA transactions with IOVAs. To perform the designed reverse-engineering steps, we first need to learn how the VAs map to IOVAs in the RNIC. This mapping is essential to create DMA requests with specific IOVAs. Besides the translation, the device also needs a *mkey*, which is used to authenticate the requestor. When a device-accessible memory is reserved, a dedicated mkey is assigned by the kernel and shared with the RNIC. An application also must initiate a DMA request to RNIC with the mkey.

Figure 1 shows the steps of the RNIC to complete an address translation from an initial VA to the target physical address (PA). Upon the kernel driver's request, the kernel reserves a device-accessible memory region with a mkey and informs it to the RNIC. This creation is performed roughly in three steps. In step 1, the kernel reserves the memory space and updates its page table (VA to PA) and I/O page table (IOVA to VA). In step 2, the kernel driver receives the VAs and IOVAs for the reserved space from the kernel. In step 3, the kernel driver delivers the VAs, IOVAs, and mkey for the reserved space to the RNIC, which keeps the information in its Memory Translation Table (MTT). When a new DMA request arrives, the RNIC looks up its MTT table to obtain the corresponding IOVA for the requested VA.

Algorithm 1: Creating a benchmark for IOTLB

Input: the number of sets *S*, the number of ways *W* and a base address of memory region on remote machine *mr_addr*
Output: none

```
procedure IOTLB_BENCHMARK(S, W, mr_addr):
    for k ← 1 to 10 000 do
        for w ← 1 to W + 1 do
            remote_addr ← mr_addr + S × w × 4KB
            RDMA_READ(remote_addr)
        end
    end
end
```

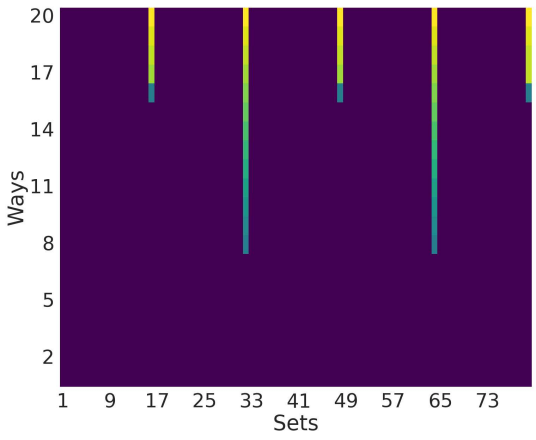


Figure 2: Heatmap depicting a result of reverse-engineering experiment on Intel Xeon Silver 4316.

3.2. IOTLB structure

Reverse-engineering microarchitectural internals of a processor generally requires hardware support [22], [37], [52], [53], [60], [70]. We also utilize hardware performance monitors to obtain necessary information of the behavior of IOTLB. Intel begins to support performance monitoring (PMON) for the IOMMU since the 3rd generation of Xeon processors, codenamed Ice Lake [27]–[30].

PMON provides a number of hardware events (event code: 0x40) for the IOMMU. Among them, three events are related to the IOTLB; `FIRST_LOOKUP` (umask: 0x1) to count the number of requests that look up the IOTLB, `MISSES` (umask: 0x20) to count the number of times that an IOTLB entry has been filled after the IOTLB miss, and `CTX_CACHE_LOOKUP` (umask: 0x40) to count the number of requests that look up a context cache after the IOTLB miss. While `CTX_CACHE_LOOKUP` counts all the IOTLB misses, `MISSES` only counts the misses that have a valid translation in a page table. We choose to use `MISSES` event as only RDMA requests that have valid addresses are used in our experiment.

We first hypothesize that the IOTLB has a set-associative structure which linearly maps IOVAs to IOTLB sets with certain associativity so that $iotlb\_set = VPN_{iova} \bmod S$, where VPN denotes a virtual page number and S the number of sets. We verify this hypothesis by utilizing a reverse-engineering technique introduced by Gras et al. [22]. First, we prepare a large memory region for RDMA requests in the server machine. Second, a client machine issues RDMA read requests to the server with varying VAs, representing different combinations of sets and ways. The VAs tested are created as in Algorithm 1, where we attempt to fill a set by set basis, filling one set first before filling another. Then, we collect `MISSES` events via PMON on the server machine for the examination.

TABLE 1: Architectural properties of IOTLB for Intel Xeon Silver 4316.

Property	4KB page	2MB page
Number of sets	32	16
Number of ways	8	8
Hash function	Linear	Linear
Indexing	IOVA[16:12]	IOVA[24:21]
Replacement policy	LRU	LRU

Figure 2 illustrates the result of our reverse-engineering efforts in a heatmap, where the brighter color indicates more IOTLB misses with the given set count (x-axis) and associativity (y-axis). The regular pattern in the figure shows that the IOTLB employs linearly-mapped mapping from IOVA to PA. Also, the figure shows the smallest way-set pair in the heatmap is 8 and 32, implying that the IOTLB has an 8-way 32-set associative cache. We later verified this conclusion independently using PMON by checking the miss events with arranged DMA transactions.

We also reverse-engineered the IOTLB's structures for 2MB pages, repeating the above test with 2MB pages. Additionally, we want to ensure whether the current IOTLB allows differently-sized pages to share the same entries. Our test was simple, priming the entire IOTLB entries with 4KB pages and attempting to evict one with a 2MB page. However, we observed that none of the primed mappings was evicted throughout this test. This result confirmed that the IOTLB employs a separate hardware component for 2MB pages with slightly different set and way counts, 16 sets and 8 ways, requiring a page's size to be the same to make distinct requests interfere with each other. Table 1 summarizes the test results on the Intel Xeon Silver 4316 processor.

3.3. Replacement policy

Another critical factor in the IOTLB structure is the replacement policy, which determines the TLB status after a miss. Prior works analyzed the replacement policy mathematically using a *permutation vector* [1], [57]. The permutation vector π_i represents an LRU order of TLB entries after a hit at the *i*-th entry. For example, if $\pi_i = (e_0, e_1, ..., e_{w-1}, e_w)$, the rightmost element e_w is the LRU, a next victim at a miss. After evicting the victim, a new

$$
\begin{aligned}
\pi_0 &= (0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7)\\
\pi_1 &= (1,\ 0,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7)\\
\pi_2 &= (2,\ 0,\ 1,\ 3,\ 4,\ 5,\ 6,\ 7)\\
\pi_3 &= (3,\ 0,\ 1,\ 2,\ 4,\ 5,\ 6,\ 7)\\
\pi_4 &= (4,\ 0,\ 1,\ 2,\ 3,\ 5,\ 6,\ 7)\\
\pi_5 &= (5,\ 0,\ 1,\ 2,\ 3,\ 4,\ 6,\ 7)\\
\pi_6 &= (6,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 7)\\
\pi_7 &= (7,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6)
\end{aligned}
$$

Figure 3: Permutation vector for perfect LRU replacement policy on IOTLB.

entry locates on the leftmost, the MRU position, and the rest of the elements shift to the right by one.

Tatar et al. [57] propose a way to investigate the permutation vector, π_i, after accessing the *i*-th entry. First, we prepare two eviction sets, $\mathcal{E}_1$ and $\mathcal{E}_2$, for a set in the IOTLB and fill the set using $\mathcal{E}_1$. Next, we access the *i*-th element of $\mathcal{E}_1$ to update the LRU order in π_i, where i is in the range of $0 \leq i \leq 7$. After the access, we examine an entry's position in π_i. This can be uncovered by counting the number of prior evictions until the target element is evicted while accessing the elements in $\mathcal{E}_2$. For example, if the target element is evicted after accessing four elements in $\mathcal{E}_2$, we can conclude that the target was in the *4*th position in π_2. Figure 3 illustrates the permutation vectors from our experiments. The figure shows all the elements keep their positions well in the LRU order in every π_i, proving that IOTLB employs a perfect LRU replacement scheme.

4. Constructing Attack Primitives

This section describes two attack primitives, DEVIOUS$_{\text{GPU}}$ and DEVIOUS$_{\text{RNIC}}$, that utilize GPU and RNIC to mount device-driven side-channel attacks on the IOTLB.

4.1. Constructing DEVIOUS$_{\text{GPU}}$

To construct the GPU-driven timing attack on IOTLB, DEVIOUS$_{\text{GPU}}$, we need a method to trigger a DMA transaction from the GPU to the host memory. When a task is offloaded into the GPU, the data needs to be migrated to the GPU's memory through DMA. Similarly, the produced results in the GPU also should be returned to the host memory after the GPU execution. The CUDA architecture thereby offers diverse memory transfer methods in its runtime API to support such data exchange between the host and GPU memory. In this work, we use a primary API, `cudaMemcpy()`, to issue a DMA transaction for GPU-to-host data transfer.

Targeting a specific IOVA with a DMA transaction requires further understanding GPU's data transfer mechanism. In CUDA, the destined host memory for the GPU-to-host data transfer must be pinned to avoid being paged out by an operating system. The CUDA driver provides *implicit* and *explicit* memory pinning. In the implicit pinning, the driver temporarily allocates pinned pages at arbitrary addresses and initiates the DMA transaction to the pages. The transferred data is then copied from the pinned to the destined memory. In this case, the location of the temporary pinned memory may change throughout multiple DMA transactions, challenging our attack that requires persistent allocated pages. The problem can be avoided by using the explicit memory pinning. CUDA provides an API `cudaMallocHost()` to allocate a persistent pinned memory at our own.

```
1  for (w=0; w<W; w++) // W: the num. of ways
2    cudaMemcpy(h_mem, d_mem,
3               bytes, cudaMemcpyDeviceToHost);
```

Listing 1: Code snippet for Prime in DEVIOUS$_{\text{GPU}}$.

Prime+Probe. The GPU driver does not allow the allocated memory to be shared between separate applications, which is necessary for *flush*-, and *evict*-based side-channel techniques. These restrictions drive the DEVIOUS$_{\text{GPU}}$ to rely on the *prime*-based attack. We thus implement a GPU-driven Prime+Probe attack for DEVIOUS$_{\text{GPU}}$, which fills and monitors the IOTLB with DMA transactions from GPUs. The attack is performed following the 3 steps after an initialization step.

In the initialization step, an attacker allocates a pinned-memory buffer for DMA transactions in the host with `cudaMallocHost()` and the GPU with `cudaMalloc()`. This allocation allows the attacker to avoid issuing a DMA transaction with the implicit pinned memory. Based on the memory addresses on the host side, the attacker constructs an eviction set $\mathcal{E}$ using the algorithm introduced by Oren et al. [50]. Note that the DMA transactions always incur in one direction from the device to the host, but not otherwise. We thus observed that only varying the memory address of the host side is enough to collect each $e \in \mathcal{E}$. After this initialization, the attacker performs the GPU-driven Prime+Probe attack following three steps.

Step 1. (*Prime*) The attacker primes a target set of the IOTLB with $\mathcal{E}$ by issuing DMA transactions for each $e \in \mathcal{E}$. Listing 1 shows a code snippet for *Prime*, where `h_mem` and `d_mem` denote the addresses of host and device memory, respectively.

Step 2. (*Wait*) The attacker waits for a while, during which a victim device that shares the IOTLB may perform a DMA transaction or not depending on the secret.

Step 3. (*Probe*) The attacker runs the code in Listing 1 again, but this time measures the execution time with a `rdtsc` instruction. DMA transactions by the victim result in a long latency. A short latency will be observed, otherwise.

4.2. Constructing DEVIOUS$_{\text{RNIC}}$

The RNIC is designed to carry out a remote DMA transaction to the remote machine over the RMDA network. InfiniBand Verbs [49], an interface for RNICs, provides useful APIs, `ibv_post_send()` and `ibv_poll_cq()`,

```
for (w=0; w<W; w++){// W: the num. of ways
    ibv_post_send(qp, ...);
    ...
    ibv_poll_cq(cq, ...);
}
```

Listing 2: Code snippet for Evict/Prime in DEVIOUS$_{\text{RNIC}}$.

to initiate DMA transactions to the RNIC and ensure its completion. With the RNIC's support, we can develop DEVIOUS$_{\text{RNIC}}$, which allows us to extend our attack to a remote timing attack. In this attack, any DMA-capable PCIe devices like GPU and NIC in the remote victim machine can be a target of the attacker, given that these devices share the IOMMU (IOTLB) with the RNIC.

Similar to the GPUs, DEVIOUS$_{\text{RNIC}}$ implements the remote side-channel using a *prime*-based technique [31], [40]. Additionally, RNIC allows a memory region on the server machine to be shared between two client applications. This enables the DEVIOUS$_{\text{RNIC}}$ to utilize an *evict*-based technique [25], which the GPU cannot. Before performing either attack, we need to complete a common initialization step, where the connection between the server's and the client's RNICs is established.

Initialization. During the initialization, the RDMA client (i.e., the attacker) first handshakes with the remote server where a victim's device locates to open an one-sided RDMA communication. Successful connection grants the attacker a mkey and VAs to the memory region of the remote server machine. Then, we prepare an eviction set $\mathcal{E}$ to the remote machine's IOTLB by utilizing the approach introduced by Oren et al. [50] again.

Evict+Reload. We first introduce *evict*-based DEVIOUS$_{\text{RNIC}}$ attack. The attacker can exploit this method when an attacker and victims share a memory region. Its details are as follows:

Step 1. (*Evict*) The attacker evicts the target set using the eviction set $\mathcal{E}$ by sending RDMA read requests for each address $e \in \mathcal{E}$, replacing all entries in the set where the target data resides. Listing 2 presents a code snippet for this step.

Step 2. (*Wait*) The attacker waits for a while, allowing time for the victim device on the server to perform a DMA transaction or not depending on the victim's secret.

Step 3. (*Reload*) The attacker runs the code in Listing 2 again, but this time measures the execution time with a `rdtsc` instruction. A latency shorter than the threshold indicates a hit in the IOTLB, confirming victim device's DMA operation.

Prime+Probe. Similar to GPUs, *prime*-based DEVIOUS$_{\text{RNIC}}$ attack follows the three steps:

Step 1. (*Prime*) Like in Evict+Reload, the attacker primes the target set with $\mathcal{E}$ using the code in Listing 2.

Step 2. (*Wait*) The attacker waits for a while, during which the victim may perform a DMA transaction or not.

Step 3. (*Probe*) The attacker probes the target set by measuring the execution time of the code in Listing 2. The slower latency than expected implies a DMA transaction has occurred by the victim.

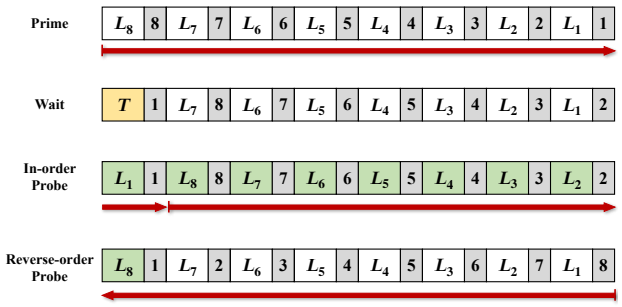


Figure 4: The difference in the number of IOTLB miss events depending on probing sequence.

TABLE 2: Experimental setup for attack evaluation.

	Sever machine	Client machine
CPU	Intel Xeon Silver 4316	AMD Ryzen 9 3950X
Motherboard	GIGABYTE MU72-SU0	MSI MAG X570S Tomahawk
GPU 1	NVIDIA Quadro K620	-
GPU 2	NVIDIA A100	
RNIC	Mellanox BlueField-2 (MLNX_OFED 5.6-1.0.3.3)	
OS	64-bit Ubuntu 20.04 LTS	

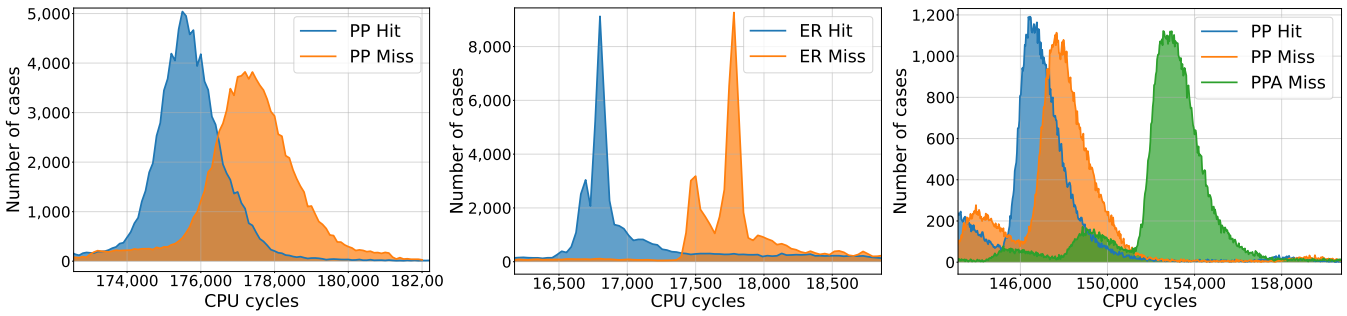
Prime+Probe with latency amplification. Unlike the local side-channel attacks, the remote side-channel attack faces one more challenge: making a robust channel that endures noises from the network. Thus, we propose a novel method to amplify the timing difference by leveraging the IOTLB's replacement policy.

Figure 4 shows the miss events with the different probing order. When the attacker primes an IOTLB set with the eviction set, the first cached entry (i.e., L_8) has the highest LRU score, 8 (gray box). Upon a victim's request for a new page (i.e., T), caused by the DMA transaction in the second step of Prime+Probe, the oldest entry L_8 is evicted from the set, and the scores for other entries are updated. At this moment, if the attacker probes the entries in the same order when they are primed, the probing experiences 8 misses (*in-order probe*). These 8 consequent misses make the victim's action more detectable with a long timing difference. Instead, if the attacker probes in the reverse order of the primed order, the probing only experiences 1 miss (*reverse-order probe*). We denote the Prime+Probe with the reverse-order probing as Prime+Probe (PP) and the Prime+Probe with the in-order probing as *Prime+Probe with Amplification* (PPA).

4.3. Evaluation and discussion

We evaluate the performance of two attack primitives under various experimental settings, and discuss the results by comparing them.

Experimental setup. The experimental setting is presented in Table 2. To test DEVIOUS$_{\text{GPU}}$, we prepared two VMs



(a) DEVIOUS$_{GPU}$ with PP (b) DEVIOUS$_{RNIC}$ with ER (c) DEVIOUS$_{RNIC}$ with PP and PPA

Figure 5: Timing distributions for two attack primitives.

on the server machine running KVM as a hypervisor. The guest OS installed on both VMs is 64-bit Ubuntu 20.04 LTS. Additionally, two GPUs, an NVIDIA A100 and an NVIDIA Quadro K620, are assigned to a VM1 and VM2, respectively, through PCI-passthrough mode.

For the DEVIOUS$_{RNIC}$, we connect three machines, two machines in Table 2 and an additional machine, over the RDMA-based network using RNICs. The two machines listed in the table still serve as an RDMA server and a client. Also, the extra machine, which has Intel Core i7-8700, and Mellanox ConnectX-6 RNIC, serves as another client. All three machines run on Ubuntu 20.04 LTS with the same device driver MLNX_OFED 5.6-1.0.3.3 for the RNICs.

Evaluation of attack primitives. To show the feasibility of the DEVIOUS$_{GPU}$ and DEVIOUS$_{RNIC}$, we design an experiment that distinguishes an IOTLB hit from an IOTLB miss based on the timing information. For this, we collect a timing information for each attack primitive. Note that we obtain the timing information for DEVIOUS$_{GPU}$ and DEVIOUS$_{RNIC}$ based on the CPU cycles with a `rdtsc` instruction. We also obtain 100,000 number of traces for each IOTLB hit and IOTLB miss case.

Table 3 and Figure 5 present an experimental result for our attack primitives. The main idea of DEVIOUS is to infer the victim device's behavior by decoding the change in the IOTLB status into the timing information. Except for DEVIOUS$_{RNIC}$ with PPA, the other attack methods experience a single IOTLB miss during the decoding. Hence, they have a similar timing difference (Δ) between hit and miss. However, if we take a deeper look into the difference, we can find out that DEVIOUS$_{GPU}$ with PP has a higher difference of about 500 CPU cycles than others. This attributes to the property of the channel that the devices construct. For instance, the RNIC-driven remote timing attacks struggle with the noise from the network that disturbs to get an apparent timing difference. However, the GPU-driven local timing attack is relatively free from the noise, through which we can obtain a clear timing information. Figure 5a, 5b, and 5c show the timing distributions collected by GPU-driven PP, RNIC-driven ER, and PP.

The property of the channel constructed from the devices also affects the attack execution time. Contrary to our expectation, the DEVIOUS$_{GPU}$ has a higher attack execution time ($\mathcal{T}$) of about 12 μs than DEVIOUS$_{RNIC}$ even if RNIC-driven timing attack is delivered from the network. This difference in the execution time comes from the characteristics of the devices. Specifically, the GPU triggers a DMA transaction with `cudaMemcpy()`. It induces an intervention of OS to perform a data transfer between a host and device memory. However, RDMA requests are directly issued and processed by the user space applications without any OS intervention. Thus, there is no data transfer between the I/O buffers (i.e., zero-copy), where RNIC directly performs DMA transactions to user space memory. Moreover, DEVIOUS$_{RNIC}$ uses one-sided RDMA requests that bypass the CPU of the remote machine, further reducing the response time.

TABLE 3: Average timing difference (Δ) and attack execution time ($\mathcal{T}$) of Evict+Reload (ER), Prime+Probe (PP) and Prime+Probe with Amplification (PPA).

Primitive	Method	Hit (cycles)	Miss (cycles)	Δ (cycles)	$\mathcal{T}$ (μs)
DEVIOUS$_{GPU}$	PP	175,633	177,275	1,642	60.42
DEVIOUS$_{RNIC}$	ER	16,798	17,891	983	52.29
	PP	146,051	147,196	1,145	45.43
	PPA		152,374	6,323	47.45

On the other hand, DEVIOUS$_{RNIC}$ with PPA has the highest 6,323 cycles of average timing difference due to its amplification. Figure 5c illustrates the RNIC-driven PPA has a more apparent timing difference than PP. However, its attack execution time is about 2 μs slower than RNIC-driven PP. Such a delay may harm the attacker due to degrading the attack performance with extended attack time. Contrarily, as discussed above, the long latency helps the attacker to discern a victim's activity in remote side-channel attacks. We observed that the noise in the network is not tiny, disturbing the attacker's accurate probing. Instead, the miss latency amplification makes the probed latency more apparent with multiple times of misses, 8 times longer in our case. This observation indicates that knowing the replacement policy in remote side-channel attacks is essential, helping the attacker build more robust attacks that bypass some glitches in its environment, such as network delays.

Comparison to DEVIOUS$_{GPU}$ and DEVIOUS$_{RNIC}$. The

IOMMU interacts with devices via the device-issued DMA transactions. Thus, the device's use case can affect how we utilize the IOTLB-based timing attack. In Section 4.1, we present DEVIOUS$_{GPU}$ to monitor another device's activities in the virtualized environment. Generally, as GPU is available in the local environment, it can be used to probe the IOTLB of the local machine that an attacker locates. Instead, the RNIC is designed to issue a DMA request to the remote machine. Thus, DEVIOUS$_{RNIC}$ can be utilized in the remote attack scenario, where an attacker monitors the remote machine's IOTLB. The use of a shared memory also can vary which kinds of attack methods an attacker uses. Contrary to RNIC, the restriction on allocating a shared memory between separate CUDA kernels forces the GPU-controlled attacker only to use *prime*-based timing attacks.

5. Cross-VM Attacks

We present cross-VM attacks using attack primitives constructed in the previous section. Our threat model assumes at least two different DMA-capable PCIe devices are connected to the PCIe network via the same root port, where the devices also share the same IOMMU. In this model, a victim VM uses one PCIe device, and an attacker VM uses another PCIe device. We also assume that the PCIe devices are dedicated to each user, so the devices cannot be shared. This constraint is common in the cloud environment, where each hardware resource is exclusively assigned to a specific VM to accommodate better isolation [30]. Therefore, the attacker cannot maliciously snoop on the victim's device. At last, this threat model does not include any software vulnerabilities (e.g., application bugs) that an unauthorized attacker can exploit.

Figure 6 shows the setting of our attacks in detail. The server machine is equipped with Intel Xeon Silver 4316 processor with 32GB RAM, and has two GPUs and one NIC attached through a PCIe switch. The attached GPU devices are NVIDIA A100 (i.e., GPU1 in Figure 6), and NVIDIA Quadro K620 (i.e., GPU2), and the NIC is NEXT-541SFP-10G. Two guests, VM1 and VM2, are running on an KVM hypervisor in the server. We use 64-bit Ubuntu 20.04 LTS Linux as the OS for both the guests and the hypervisor.

We consider two cross-VM attack scenarios. In the first attack (in Section 5.1), two isolated virtual machines from different domains construct a covert channel by using dedicated GPUs, which are assigned to each VM with a PCI-passthrough mode. In the second attack (in Section 5.2), a spy VM delivers SSH keystroke timing attack on a victim VM. The spy utilizes his/her own GPU assigned with a PCI-passthrough mode to probe the status of IOTLB made by the victim's NIC.

5.1. Attack 1 - Cross-VM covert channel

The VMs in this test run on the same machine, but they are strictly isolated and have no means to access the other VM's memory. However, we construct a prohibited covert channel through the IOTLB and pass messages between two VMs. Note that this new channel bypasses all prior software/hardware protections against the CPU cache-based covert channels [18], [23], [39], [56], [74].

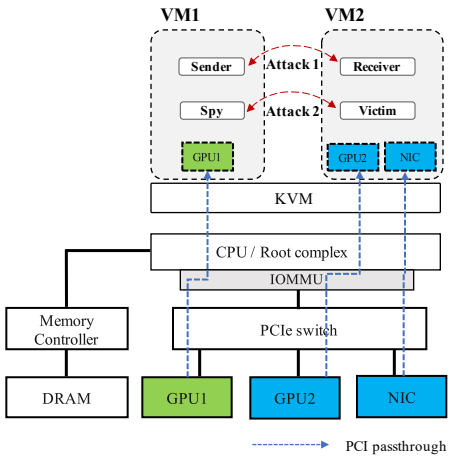


Figure 6: Cross-VM attacks on IOTLB.

Constructing a covert channel. Before constructing an IOTLB-based cover channel, the sender and the receiver must arrange their communication protocol. The sender and the receiver must determine an IOTLB set, $\mathcal{S}$, which they use for the channel. Also, they agree that the sender posts 1 bit every fixed time window. In this test, the sender post '1' by priming whole entries in $\mathcal{S}$ using the GPU, wiping out any last changes. Otherwise, the sender post '0' by staying inactive during the time window. During the next window, the receiver retrieves the posted 1 bit by probing any changes in $\mathcal{S}$. A longer-than-expected probing latency indicates '1'; otherwise, '0'.

Evaluation. The performance of the covert channel is evaluated under the same experimental setting described in Section 4.1. In the experiment, we use 100Kb data of bit sequence, where bits '1' and '0' are uniformly distributed randomly. The sender and the receiver synchronize with the pre-determined time window, whose size affects a covert channel's effectiveness, such as throughput or bit-error rate. The experiment reveals that, on average, the sender needs 76 μs to *prime* an IOTLB set with the GPU. Considering this result, we test by altering the time window size from 100 μs to 400 μs.

Figure 7 shows the throughput and the bit error rate (BER) from our covert channel evaluation with a single IOTLB set. As expected, we get lower BER and smaller channel bandwidth as the time window size grows. The covert channel with the single IOTLB set shows 2.54 Kbps throughput and 2.18% BER with a time window of 250 μs. This channel bandwidth can be enhanced by using multiple sets. We confirm that the covert channel with 16 IOTLB sets results in 40.64 Kbps throughput. Also, various enhancement techniques can be employed for further improvement [2], [24], [48].

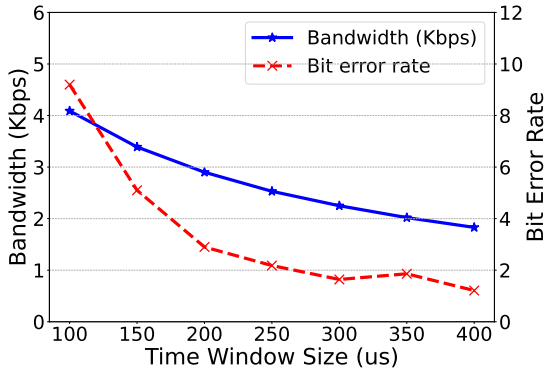


Figure 7: Performance of a cross-VM covert channel over a single IOTLB set.

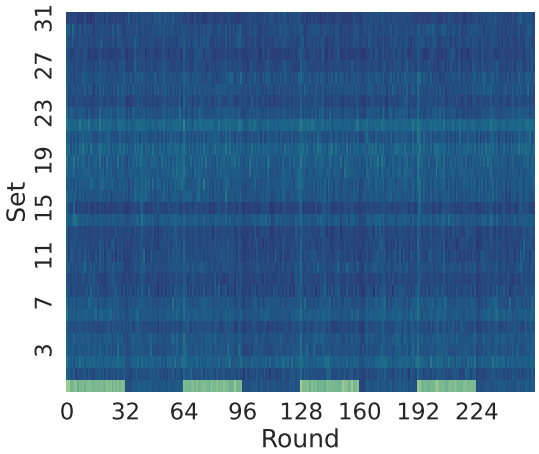


Figure 8: Heatmap graph for identifying the RX ring buffer.

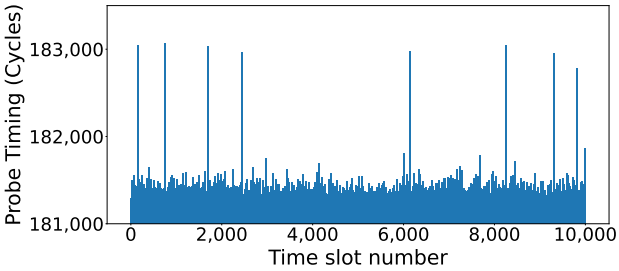


Figure 9: IOTLB trace for arrivals of SSH packets.

5.2. Attack 2 - Keystroke timing attack

The SSH protocol is subject to keystroke timing attacks [38], [61], [80], as individual transmitted SSH packets are directly related to keystroke events from the SSH client. The interarrival time of packets reveals the information on keydown-to-keydown time of users. Previous studies relevant to our work rely on the contention on a last level cache [38], [65] in server-grade processors equipped with Data-Direct I/O (DDIO), or the congestion of a PCIe link [64] as a side-channel to leak the keystroke information. In this attack, we use the IOMMU as a novel leakage source of the timing information of packet's arrival at the victim's NIC.

We consider a spy who mounts SSH keystroke timing attack with DEVIOUS$_{\text{GPU}}$. The spy is located in a virtual machine VM1, as shown in Figure 6, and uses a dedicated GPU (i.e., GPU1) to probe IOTLB shared with other devices. The victim is located in another virtual machine VM2, where a dedicated NIC device is assigned with a PCI-passthrough mode. The victim is running an SSH server, and accepts SSH connections from remote clients connected through the Ethernet NIC device. For this attack, the spy does not need any SSH connection or access permission to VM2, where an SSH server is running. The attack can be performed as long as the spy and the victim share the same IOTLB.

In Linux, an RX ring buffer is a list of packet descriptors with head and tail pointers pointing out the first and last element of the list [76]. Each descriptor contains a physical address of the receive buffer for a received packet and its size. When a packet arrives, the NIC first looks for an available descriptor in the RX ring, and then transfers the received packet to the receive buffer specified by the descriptor.

The main idea of our attack originates from the fact that the NIC device performs DMA access to the RX ring buffer in the physical memory upon the receipt of packets. The DMA access to the ring buffer will consequently cause an IOTLB lookup, which can be observed by our IOTLB probing techniques.

Identifying IOTLB sets. The first step for our attack is to identify IOTLB sets associated with the ring buffer. Linux allocates a 4KB page for the RX ring buffer by default, and its DMA address is mapped with consistent/coherent DMA type as the ring buffer remains throughout packet transmissions. Hence, our hypothesis is that the cache activity will be consistently observed in only one set of the IOTLB while receiving packets.

We conduct an experiment to verify our hypothesis, and identify the associated IOTLB set. In the experiment, we probe cache activities for the entire sets of IOTLB while repeatedly sending packets from a remote machine to the NIC assigned to VM2. We command a program in the remote machine to perform 256 rounds of a packet-sending operation at a fixed time interval of 146 μs. In each round of receiving a packet, we use DEVIOUS$_{\text{GPU}}$ to measure the cache activity for the entire IOTLB sets. To make it more clear, we control the remote program so that every 64 rounds, it keeps sending packets only for the first 32 rounds. Figure 8 presents the experimental result. The heatmap shows cache activities of each set for the 256 rounds, where the bright color implies more IOTLB misses. The distinguishable pattern is only observed in the first set (i.e., the set number 0), indicating that the RX ring buffer is located at the set 0. We confirm this result remains consistent throughout multiple system reboots and guest resets.

TABLE 4: Evaluation result of keystroke timing attack.

	$\Delta_t(=\lvert\mathcal{T}_T-\mathcal{T}_I\rvert)$	$\Delta_g(=\lvert\mathcal{T}_G-\mathcal{T}_I\rvert)$
Mean	0.81 ms	2.87 ms
Std. deviation	10.07	10.08

Inferring packet arrival. The spy uses the information about the IOTLB set associated with the RX ring buffer to infer a packet's arrival at the NIC. That is, he/she repeatedly probes the set 0 by using GPU1 assigned to VM1, and traces the probe results to obtain the timing information of received packets.

Figure 9 is an example of a trace obtained from probing the IOTLB set with the GPU-based probing technique while a word `'notebook'` is being sent to the SSH server. Each SSH packet is created and transmitted to the server whenever a single character is input from the SSH client. When an SSH packet arrives, the NIC performs DMA access to the RX ring buffer, and a lookup to the IOTLB set 0 will take place. The lookup by the NIC will cause an eviction of entries primed by the spy. As a result, the spy observes IOTLB miss, which is shown as a peak in the trace as shown in Figure 9. By analyzing the peaks, the spy is able to obtain the interarrival time of the received packets, which provides timing information that enables the spy to deduce the key pressed by the SSH client.

Evaluation. We evaluate the performance of our keystroke timing attack. Our experimental environment is the same as the setting presented in Figure 6. It is noteworthy that the objective of our attack is not to completely recover the characters, but rather to obtain the interarrival time of the received packets using the probing technique. In the evaluation, we measure the accuracy of detecting keystroke events by IOTLB probing, i.e., how much difference in the timing is between the detection and ground truth?

In the experiment, we run a remote SSH client connected through the NIC to the SSH server running in VM2. The SSH client is automated to send 1,000 words of a dictionary to the SSH server. As the time interval of keystrokes by human is observed to be around 240 ms on average [36], the timing of sending a packet (i.e., an individual character) is determined by emulating the human behavior.

The SSH client records a timestamp whenever a character is input, which is used as a ground truth in the evaluation. We also trace the received packets through the NIC by using tcpdump, and use the timestamps of the packet arrival for comparing our result. While packets are being sent from the SSH client, the spy performs tracing the IOTLB set 0 with the GPU-based probing technique. After the trace has been obtained, we figure out timestamps of peaks, like shown in Figure 9, from the trace.

Table 4 presents statistical differences of timestamps captured by the GPU-based probing technique from ground truth and tcpdump. $\mathcal{T}_I$ denotes a timestamp with a peak in the IOTLB trace for a certain character input from the SSH client. $\mathcal{T}_G$ and $\mathcal{T}_T$ denote a timestamp obtained from ground truth and tcpdump for the same character input, respectively.

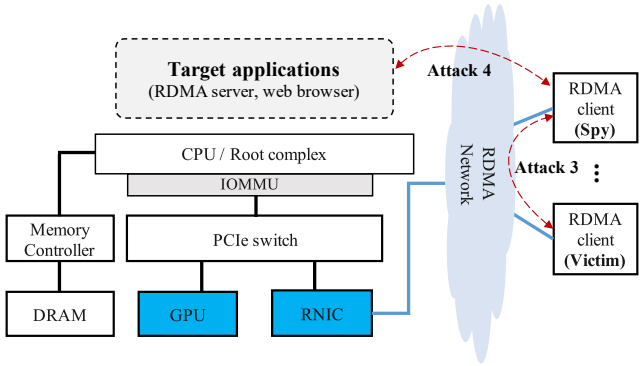


Figure 10: Remote timing attacks on IOTLB.

We observe from the result that the timing difference (Δ_t) with tcpdump is much smaller than the difference (Δ_g) with the ground truth. We also observe that the difference Δ_t is less than 1 ms, which is significantly small compared to 240 ms of the average interarrival time. From these observations, we come to a conclusion that our keystroke timing attack is as accurate as sniffing-based timing attacks [61], and can effectively recover keystroke events with high probability.

6. Remote Timing Attacks

In this section, we present remote IOTLB-based timing attacks. We consider two attack scenarios, where a spy and victim are located on different machines connected over an RDMA network. Figure 10 illustrates the setting of our remote timing attacks. In the threat model, the spy has control over an RDMA client, which is connected to a remote machine where target applications such as an RDMA server or web browser are running. However, the spy has neither direct control over the target machine, nor other RDMA clients.

In the first attack (in Section 6.1), a spy delivers the DEVIOUS$_{\text{RNIC}}$ attack to a target machine running Apache Crail, a key-value store. The spy tries to infer the access pattern to specific key-value pairs of a victim on other RDMA client. In the second attack (in Section 6.2), a spy mounts website fingerprinting against the remote target machine running a web browser.

It is noteworthy that remote attacks are also available in the cross-VM setting, as shown in Section 5. For example, Attack 4 (the second attack) can be carried out when one VM has an RDMA application that connects to a remote adversary while the other VM (the victim) is using a web browser. Additionally, Attack 3 (the first attack) is also viable in a cross-VM attack scenario like this.

6.1. Attack 3 - Attacking Apache Crail

We consider a spy trying to infer access patterns of a key-value store. The spy is located in an RDMA client, connected to the server over the RDMA network, and uses DEVIOUS$_{\text{RNIC}}$ to deliver the attack.

The server runs Apache Crail for the key-value store. Crail is a high performance distributed storage [5], [63], consisting of a name node and data node[1]. The data node has a storage, which is a large list of 1 MB data blocks, to keep up user data. The name node contains all the metadata related to the user data stored in the data node. In the RDMA-based configuration, separate memory regions are allocated for the name node and data node. The name node serves a client's read request for a key-value item. It retrieves metadata such as a location of the corresponding value in the data node. As the metadata is stored in a memory region, such retrieval induces an RDMA read, which internally makes a DMA access to the memory region.

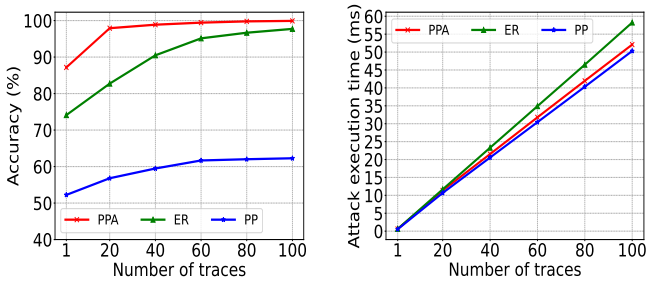
The spy is interested in the access pattern of a victim client, i.e., whether the victim has requested a certain key-value item or not. Thus, the spy probes the IOTLB sets corresponding to the memory region of the metadata in the name node to leak the access pattern. In this work, we consider two attack scenarios according to the setting of the memory region of the name node, i.e., the memory region is shared with the spy or not.

Attack with the separate memory region. In this attack, we suppose that the spy acts as a client of the key-value store, and has access to a different memory region on the server. As the spy has no memory regions shared with the victim on the RDMA server, there are no options but to use *prime*-based techniques such as PP and PPA in the DEVIOUS$_{\text{RNIC}}$ attack. Hence, the first step for the attack is to construct an eviction set that occupies the same IOTLB set of the victim's target memory region. As a name node in the Crail has memory regions allocated with hugepage (i.e., 2MB page), the eviction set needs to be constructed upon the IOTLB of 16 sets for hugepages. The next step is to identify the set of the memory region where the target key-value item resides. It can be easily achieved by investigating the entire IOTLB sets and looking for the one showing expected usage patterns of key-value items.

Attack with the shared memory region. In this scenario, the spy has access to the memory region of the name node shared with the victim client. For instance, this is the case where the spy acts as an YCSB Crail client [17], which is one of the Crail applications that support memory sharing. As the spy has the shared memory region with the metadata for the target key-value item, DEVIOIS$_{\text{RNIC}}$ with ER is available in this attack.

Evaluation. The experimental setting for the attack evaluation is the same as that depicted in Figure 10. For the experiment, we implement a spy program that performs DEVIOUS$_{\text{RNIC}}$ with PP, PPA, and ER methods to a given eviction set or an address of the memory region. To identify the target set, we search for IOTLB sets that exhibit significant activity when sending requests to key-value pairs. We measure the performance of the attacks in terms of the accuracy and execution time by running the spy program while a victim client is sending a read request of a key-value item to the Crail server. We allow the program to run multiple times to obtain multiple traces of the IOTLB for the same read request. The number of traces (N_t) reflects the attacker's ability. That is, traces more than once (i.e., $N_t > 1$) imply that the attacker can control the victim to send multiple requests of the target key-value item while a single trace (i.e., $N_t = 1$) implies no control over the victim. Even though having $N_t > 1$ may not be a practical scenario in Crail, we assume it to evaluate the overall attack performance in relation to their capability (N_t).

1. The Apache Crail project retired on June 2022.



(a) Attack accuracy (b) Attack execution time

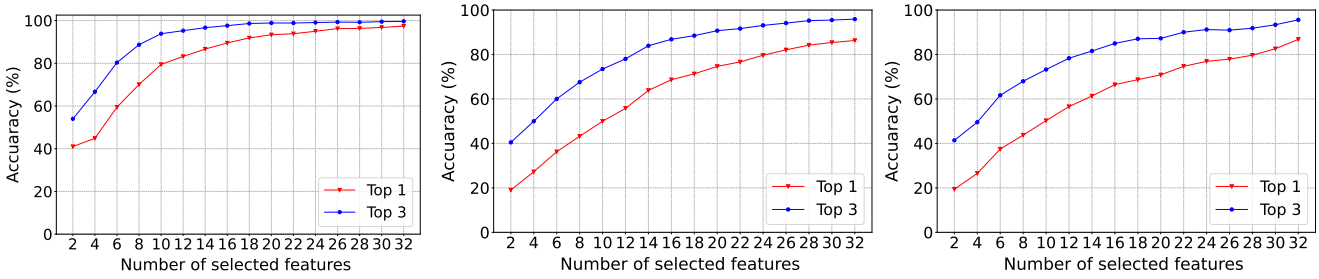
Figure 11: Evaluation of DEVIOUS$_{\text{RNIC}}$ on Apache Crail.

Figure 11 shows the performance of the attacks according to the number of traces. The more we collect a large amount of trace, the more the attack accuracy and attack execution time increase. As shown in Figure 11a, the accuracy of PPA sharply rises from 89.47% to 99.18% with $N_t = 20$ while the accuracy of PP has stabilized to 62.66% even with $N_t = 100$. We attribute this to the low accuracy of the PP method. Figure 11b shows attack execution times for each method. It is noteworthy that ER shows the highest attack execution time. This has to do with an implementation method of each attack. Specifically, nine individual RDMA read requests are used for the ER attack in our implementation; eight requests to evict a set, and one request for reloading. On the other hand, PP and PPA attack needs only eight RDMA requests, as the requests in the *Probe* step are the same as the requests in the *Prime* step. Hence, the one more RDMA request of the ER method results in the highest attack execution time.

6.2. Attack 4 - Website fingerprinting

We consider a spy mounting a website fingerprinting attack. As shown in Figure 10, the spy is located at a remote RDMA client, which is connected to a victim machine over the RDMA network. The spy tries to infer websites that the victim visits by probing the IOTLB status with DEVIOUS$_{\text{RNIC}}$. Our attack exploits the GPU acceleration of modern web browsers. There are generally two types of GPUs available in computers: integrated and discrete GPUs. Our attack is not affected by the type of GPU since both are managed by IOMMU [44], [82]. Nevertheless, we primarily focus on discrete GPUs since they are more widely used.

Rendering a web page. As the internal rendering process varies with web browsers, we focus on Chrome in this work. In Chrome, a process called renderer is responsible



(a) $N = 100$ (b) $N = 200$ (c) $N = 300$

Figure 12: The classification accuracy regarding the number of selected features (N denotes the number of websites.).

for rendering a web page in each tab. It translates HTML, JavaScript, and CSS into a Data Object Model (DOM) object and a computed style for the DOM object. The renderer then calculates the geometry of the page and determines the paint order. Once the information necessary for displaying parts of the page has been gathered, the renderer performs rasterizing, i.e., converting the rendering information into a bitmap. The resulted bitmap is then transferred to the framebuffer.

Inferring website. Chrome turns on the GPU acceleration by default, and offloads some parts of the rendering operation, such as rasterization and compositing, to the GPU. To further speed up the acceleration, offloading takes place as soon as necessary elements become available.

The GPU offloading induces a DMA transfer to the device memory, creating unique DMA access pattern according to the content and shape of the web page. This unique pattern is also observed in a trace of the IOTLB status. Figure 14 in Appendix D shows examples of traces for the entire IOTLB sets obtained during visiting websites. The traces are obtained by using DEVIOUS$_{\text{RNIC}}$ with the PPA technique. On average, it takes 600ms to load a web page, but a single execution of PPA for the entire IOTLB takes less than 2ms. Therefore, DEVIOUS$_{\text{RNIC}}$ can collect over 300 IOTLB traces, which is enough to capture unique and identifiable traces for each website. With this trace, the spy can infer the information about the web site visited by the victim.

Building classifier. To identify a web site from the obtained IOTLB trace, we build a deep learning-based classification model. As the trace is time-series data, we choose a bidirectional LSTM model (Bi-LSTM). We stack up the attention layer for improving the weight per label to the output layer of Bi-LSTM. As the last layer of the classifier, the Att-BLSTM model constructed the dense layer that contains a softmax activation function to classify time series data.

We alleviate an overfitting problem by adding a dropout layer and batch-normalization layer to the output of each classifying layer. We also apply Early-Stop method to dynamically decrease the learning rate if validation loss does not increase after the n iterations, where n is set by the patience argument. Hyper-parameters for our model are shown in Table 5.

TABLE 5: Hyper-parameters for the classification model (N denotes the number of websites in the closed set $\mathcal{S}$.).

Hyper-parameter	Value
Size of hidden layer	96
Size of attention layer	96
Size of dense layer (N=100,200,300)	100,200,300
Dropout	0.5
Learning rate	0.013
Learning rate decay	0.19
Batch size	32
Epoch	200

To obtain the training data, we choose top 300 websites from Alexa Top 1M Sites [3]. Then, we collect traces of the entire IOTLB sets for each website while the web page is being displayed in the Chrome web browser. The data is divided into a training set and a test set with a ratio of 8:2. The data is also normalized to make the classifier resilient against specific noise such as outlier.

Evaluation. We evaluate the accuracy of our classification model under the experimental setting depicted in Figure 10. We use Chrome on Linux with version 103.0.5060.114 as a victim's web browser. The experiment is conducted under a default setting of the GPU acceleration in the web browser. The evaluation follows a closed-world scenario: the victim is restricted to visit websites in a closed set $\mathcal{S}$.

The classification accuracy is presented in Table 6. For instance, the probability that the visited website matches the topmost output of our classification model under the closed set $\mathcal{S}$ of 100 websites is 98.90%. The probability that the website is in the top 3 ranked outputs under the same closed set is 99.82%.

The classification model is built with features of the entire IOTLB sets. Our expectation is that not all the features (i.e., the sets) are necessary for accurate classification of websites. Hence, we also evaluate our model with respect to the number of selected features. For this, we rank each feature based on its importance using a random forest feature selection algorithm, and choose features with the highest ranks in the evaluation.

Figure 12 shows the evaluation result. In the case of classifying among 100 websites (i.e., $N = 100$), the accuracy

TABLE 6: Accuracy of classifying websites.

	N=100	N=200	N=300
Top 1 Acc.	98.90%	88.57%	86.75%
Top 3 Acc.	99.82%	96.77%	95.64%

for Top 1 and Top 3 is close to 90% even with only a half of 32 features selected. This result implies that the spy can achieve high accuracy even by probing just a small portion of IOTLB sets, enabling faster attacks that are effective in fingerprinting websites with short loading time.

There are several limitations of the demonstrated attack. First, the generality of the classifier is limited because the IOTLB footprints vary depending on the specific GPU model or version of the web browser, thus requiring the classifier to be built accordingly. Second, the attack is susceptible to noise from other devices that share the same IOTLB, lowering the classifier's accuracy (See Appendix A.1). Finally, as for GPU rendering using huge pages, the accuracy will decrease as the attacker is only able to obtain coarser-grained information (See Appendix A.2).

7. Mitigation

In this section, we discuss some possible mitigations against the DEVIOUS attack.

Anomaly detection. Anomaly-based detection of DEVIOUS can be an effective mitigation strategy. Similar to previous detection techniques for CPU-based side-channel attacks [7], [15], [46], [81], the hardware events can be utilized to develop a runtime detection method for our attack. Although there already exists a number of well-established detection methods for the CPU-based side-channel attacks, we cannot directly use them for our attack. This is because they utilize hardware events related to the behavior of CPU caches or TLBs, while DEVIOUS never affects these hardware. Thus, it is necessary to figure out the relevant hardware events for the detection of our attack.

Fortunately, Intel provides some hardware events related to the IOTLB since the 3rd generation of Xeon processors, as described in Section 3.2. To detect anomalies, we suggest utilizing four events related to IOTLB: FIRST_LOOKUPS (umask: 0x1), 4K_HITS (umask: 0x4), 2M_HITS (umask: 0x8), and MISSES (umask: 0x20). These events are crucial for observing the IOTLB's state in regards to all DMA transactions. Detecting DEVIOUS follows a similar approach to that of detecting CPU-based side-channel attacks [7], [15], [46], [81], with the exception being the hardware events utilized. As a result, the implementation of the detector is uncomplicated. According to our assessment, the detector incurs a runtime overhead of less than 5%. Further information on the implementation and evaluation of the detector is presented in Appendix B.

IOTLB partitioning. Sharing the IOTLB among multiple devices becomes the main cause of the security risk. Hence, the IOTLB partitioning can be another effective mitigation strategy. With this feature, only devices that belong to the same protection domain share a portion of the IOTLB. The IOTLB partitioning fundamentally eliminates any possibilities of the contention with devices from different domains on the IOTLB. Unfortunately, we have no idea of any works in progress to redesign the IOTLB for this purpose. We only found an Intel patent that proposes a single partitioning approach for the IOTLB [26]. It is to prioritize each Process Address Space ID (PASID) and determine the maximum number of IOTLB entries to assign for each device depending on the priority. We expect that this approach can mitigate the DEVIOUS attack. However, the partitioning approach obstructs devices from fully utilizing the IOTLB entries. This will lead to a higher occurrence of IOTLB misses, negatively impacting the I/O performance of the system. According to the previous results of partitioning-based mitigation approaches for side-channel attacks, L3 cache partitioning resulted in a runtime performance overhead of 5.9% [39], due to the increased miss rate. Regarding TLB partitioning, evaluation through SPEC benchmarking showed a threefold increase in the number of TLB misses [18].

Randomization. Randomizing IOTLB is a promising hardware-based mitigation approach against DEVIOUS. The basic idea is to randomize the IOTLB set when allocating an entry in the case of a miss. This approach has been demonstrated by Deng et al. [18] for TLBs, but can also be applied to the IOTLB. Specifically, a hit operates the same as a typical IOTLB, but upon a miss, the hardware fills an IOTLB entry with a random address translation instead of the desired one. The actual IOTLB set of the allocated entry is determined not only by the set index bits but also by a secret. Without knowledge of the secret, attackers cannot construct eviction sets or identify target set, which is a necessary step to deliver attacks. In terms of the performance overhead, Deng et al. reported a 7% increase in the area of the randomized TLB implementation [18].

Application-level mitigation. As fixing the IOTLB hardware requires a considerable amount of time, we suggest short-term measures that can be implemented in applications. To address Attack 2, the SSH client can inject dummy packets [59] or introduce random delays [61] to disrupt the inference of keystroke timing based on the observed interarrival time of received packets. When it comes to Attack 3, it is crucial to conceal the victim's access pattern to the key-value store. This can be achieved through the use of ORAM (Oblivious RAM) [21], which ensures access pattern privacy. Some practical ORAM implementations exhibit 1.0 - 4.7 slowdown in the storage performance [9], showing the feasibility of ORAM. To mitigate Attack 4, web browsers can introduce dummy DMA transactions to the rendering process to prevent an attacker from inferring the website (See Appendix C for further discussion on the mitigation).

8. Related Work

Attacks to the IOMMU. There are several works that studied the vulnerabilities of IOMMU. Morgan et al. [44],

[45] discovered that there is a vulnerable time window between the system boot and the IOMMU initialization, during which an attacker can initiate malicious DMA requests that compromise the IOMMU configuration data, such as a page table for DMA remapping, stored in the physical memory. This allows an attacker to bypass the IOMMU protection, enabling DMA attacks. Zhu et al. [82] exploit that the IOTLB is not kept coherent with the IOMMU page table. This opens another vulnerability window, through which an attacker can manipulate a page table entry. Markettos et al. [42] extensively investigated the IOMMU usage of modern operating systems by using an FPGA-based analysis platform named Thunderclap. The investigation reveals inadequate use of the IOMMU in certain OSes like macOS and Linux, allowing attackers to bypass the IOMMU.

Previous works described above mainly focus on vulnerabilities due to the misconfiguration and misuse of the IOMMU. These vulnerabilities reopen DMA attacks [6], [12], [62], circumventing the IOMMU-based protection. Unlike the previous studies, our work uncovers a new side-channel vulnerability, introduced by a shared caching structure inside the IOMMU. A DEVIOUS attacker can leak the confidential information through the side-channel without having direct access to the host memory.

Side-channel attacks on peripherals. Side-channel attacks are evolving from targeting software via CPU-based side-channel [24], [25], [31], [40], [79] to targeting peripheral devices exploiting a variety of shared hardware resources as a side-channel source. In Pythia [69], an attacker utilizes the RNIC's SRAM that caches metadata for the RDMA communication as a side-channel, and constructs an *evict*-based remote timing attack over the RDMA network. In NetCAT [38] and Packet Chasing [65], a last-level cache (LLC) is utilized as a side-channel source. With the DDIO technology, a victim device has direct access to the LLC bypassing the physical memory. Thus, an attacker can leak a victim's secret by probing the LLC. In InvisiProbe [64], an attacker exploits the congestion from a PCIe switch as a leakage source, and constructs a side-channel attack to leak secret of the victim device.

Similar to our work, IOTLB-SC [67] proposes an attack that targets the IOTLB. However, this work differs in two key respects. First, we have done a thorough reverse engineering of the IOTLB's internal structure. The newly discovered knowledge of IOTLB enables more powerful attacks, such as PPA (in Section 4.2). Second, unlike our work, they only use FPGA as an attack device, limiting its generality. However, we demonstrated multiple attack primitives using different PCIe devices in cross-VM and remote attack scenarios.

Reverse-engineering (IO)TLB structure. An address translation cache has recently been getting attention as a new source of side-channel attacks [22], [37], [57]. Compared to CPU caches, the most internals of TLBs are unknown, requiring reverse-engineering efforts. The authors of TLBleed [22] and TagBleed [37] conducted reverse-engineering on the Intel processor's TLB architecture with 4KB, 2MB, and 1GB pages by utilizing hardware performance counters. Tatar et al. [57] uses a TLB desynchronization technique for reverse-engineering, and succeeded in disclosing the replacement policy of Intel's TLB. On the other hand, there was an effort to figure out an architectural property of IOTLB. Peglow [51] relied on timing measurements to reverse-engineer the microarchitectural internals of IOTLB, but failed to completely disclose it. Unlike the previous work, we rely on a hardware performance monitoring unit for the uncore part of Intel processors and successfully reverse-engineer the internals of IOTLB including its cache organization and replacement policy.

9. Conclusion

The IOMMU is introduced to tackle security problems caused by malicious DMA-capable devices. With the address translation capability, the IOMMU creates a virtualized and isolated DMA address space for each device, achieving device isolation on the basis of protection domains. However, the IOTLB, an address translation cache inside the IOMMU, can be abused by attackers to deliver side-channel attacks. In this paper, we performed a deep dive into undisclosed microarchitectural internals of the IOTLB by reverse-engineering an Intel processor, and presented DEVIOUS, a novel device-driven side-channel attacks. Our attack employs two DMA-capable PCIe devices, a GPU and an RNIC, to generate device-initiated memory requests that create contention with a victim's device on the IOTLB. It allows attackers to infer the victim's secret information through the IOTLB side-channel with high fidelity. We argued the security impact of DEVIOUS by demonstrating practical attacks under the real-world setting like a highly-isolated virtualized environment and the network of RDMA machines with IOMMU-based protections applied. With security solutions proposed in this paper, we believe that DEVIOUS can be effectively mitigated.

Responsible disclosure. We reported our attacks to both the Google Chrome security team and the OpenSSH maintainer on March 20, 2023. At the time of finalizing this paper, we have not yet received any responses from either of them.

Acknowledgments

We would like to thank our reviewers for their valuable feedback to improve our paper. This work was supported by IITP grant (IITP-2023-2020-0-01819, IITP-2023-2021-0-01810) and NRF grant funded by the Korea government (No.2023R1A2C2006862, NRF-2021R1A6A1A13044830). This work was supported by NSF CAREER Award CCF-2146475.

References

[1] A. Abel and J. Reineke, "Measurement-based modeling of the cache replacement policy," in *RTAS*, 2013, pp. 65–74.

[2] J. Ahn, J. Kim, H. Kasan, L. Delshadtehrani, W. Song, A. Joshi, and J. Kim, "Network-on-chip microarchitecture-based covert channel in gpus," in *MICRO*, 2021, pp. 565–577.

[3] Alexa, "Alexa Top Websites," 2022, accessed on 26-09-2022. [Online]. Available: https://www.expireddomains.net/alexa-top-websites/

[4] AMD, "Amd i/o virtualization technology (iommu) specification," 2021. [Online]. Available: https://www.amd.com/system/files/TechDocs/48882_IOMMU.pdf

[5] Apache, "Crail: High-performance distributed data store," https://crail.incubator.apache.org/, 2018, accessed on 26-09-2022.

[6] D. Aumaitre and C. Devine, "Subverting windows 7 x64 kernel with dma attacks," *in HITBSecConf*, 2010.

[7] M.-M. Bazm, T. Sautereau, M. Lacoste, M. Sudholt, and J.-M. Menaud, "Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters," in *FMEC*, 2018, pp. 7–12.

[8] M. Becher, M. Dornseif, and C. N. Klein, "Firewire: all your memory are belong to us," *in CanSecWest*, 2005.

[9] V. Bindschaedler, M. Naveed, X. Pan, X. Wang, and Y. Huang, "Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward," in *ACM CCS*, 2015, pp. 837–849.

[10] E.-O. Blass and W. Robertson, "Tresor-hunt: Attacking cpu-bound encryption," in *ACSAC*, 2012, pp. 71–78.

[11] A. Boileau, "Hit by a bus: Physical access attacks with firewire," *in Ruxcon*, 2006.

[12] R. Breuk and A. Spruyt, "Integrating dma attacks in exploitation frameworks," *University of Amsterdam, Tech. Rep.*, 2012.

[13] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "Reload+refresh: Abusing cache replacement policies to perform stealthy cache attacks," in *USENIX Security Symposium*, 2020, pp. 1967–1984.

[14] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A decoupled, client-driven key-value store using rdma," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3537–3552, 2017.

[15] J. Cho, T. Kim, S. Kim, M. Im, T. Kim, and Y. Shin, "Real-time detection for cache side channel attack using performance counter monitor," *Applied Sciences*, 2020.

[16] Chrome, "Analyze runtime performance." [Online]. Available: https://developer.chrome.com/docs/devtools/performance/

[17] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, 2010, pp. 143–154.

[18] S. Deng, W. Xiong, and J. Szefer, "Secure tlbs," in *ISCA*, 2019, pp. 346–359.

[19] U. Frisk, "Direct memory attack the kernel," *in DEFCON*, 2016.

[20] Y. Gao, Q. Li, L. Tang, Y. Xi, P. Zhang, W. Peng, B. Li, Y. Wu, S. Liu, L. Yan *et al.*, "When cloud storage meets RDMA," in *NSDI*, 2021, pp. 519–533.

[21] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *J. ACM*, vol. 43, no. 3, p. 431–473, 1996.

[22] B. Gras, K. Razavi, H. Bos, and C. Giuffrida, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *USENIX Security Symposium*, 2018, pp. 955–972.

[23] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa, "Strong and efficient cache side-channel protection using hardware transactional memory," in *USENIX Security Symposium*, 2017, pp. 217–233.

[24] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, 2016, pp. 279–299.

[25] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *USENIX Security Symposium*, 2015, pp. 897–912.

[26] K. Guo, W. Li, J. Wang, L. Ma, M. Lukoshkov, and Y. Huo, "Address translation technologies," Nov. 26 2020, US Patent 11,422,944.

[27] Intel, "Intel® xeon® processor scalable memory family uncore performance monitoring," 2017. [Online]. Available: https://kib.kiev.ua/x86docs/Intel/PerfMon/336274-001.pdf

[28] Intel, "Utilizing the intel® xeon® processor scalabe family iio performance monitoring event," 2019. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/utilizing-the-intel-xeon-processor-scalable-family-iio-performance-monitoring-events.html

[29] Intel, "3rd gen intel® xeon® processor scalable family, codename ice lake, uncore performance monitoring," 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/download/3rd-gen-intel-xeon-processor-scalable-uncore-pm.html

[30] Intel, "Intel® virtualization technology for directed i/o," 2022. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/671081/intel-virtualization-technology-for-directed-i-o-architecture-specification.html

[31] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$A: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *IEEE S&P*, 2015, pp. 591–604.

[32] N. S. Islam, D. Shankar, X. Lu, M. Wasi-Ur-Rahman, and D. K. Panda, "Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store," in *ICPP*, 2015, pp. 280–289.

[33] D. Jaeger, K.-F. Krentz, M. Richly, C. Willems, W. Dawoud, and I. Takouna, "Xen episode iv: The guests still strike back," *in Cloud Computing Security Summer Term*, 2011.

[34] T. M. John, "Privacy leakage via write-access patterns to the main memory," *Master's Theses*, 2017. [Online]. Available: https://opencommons.uconn.edu/gs_theses/1134/

[35] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using rdma efficiently for key-value services," in *ACM SIGCOMM*, 2014, pp. 295–306.

[36] K. S. Killourhy and R. A. Maxion, "Free vs. transcribed text for keystroke-dynamics evaluations," in *LASER*, 2012, pp. 1–8.

[37] J. Koschel, C. Giuffrida, H. Bos, and K. Razavi, "Tagbleed: breaking kaslr on the isolated kernel address space using tagged tlbs," in *EuroS&P*, 2020, pp. 309–321.

[38] M. Kurth, B. Gras, D. Andriesse, C. Giuffrida, H. Bos, and K. Razavi, "Netcat: Practical cache attacks from the network," in *IEEE S&P*, 2020, pp. 20–38.

[39] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *HPCA*, 2016, pp. 406–418.

[40] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE S&P*, 2015, pp. 605–622.

[41] X. Lu, M. W. U. Rahman, N. Islam, D. Shankar, and D. K. Panda, "Accelerating spark with rdma for big data processing: Early experiences," in *HOTI*, 2014, pp. 9–16.

[42] T. Markettos, C. Rothwell, B. F. Gutstein, A. Pearce, P. G. Neumann, S. Moore, and R. Watson, "Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals," in *NDSS*, 2019.

[43] I. Messadi, S. Neumann, N. Weichbrodt, L. Almstedt, M. Mahhouk, and R. Kapitza, "Precursor: a fast, client-centric and trusted key-value store using rdma and intel sgx," in *Middleware*, 2021, pp. 1–13.

[44] B. Morgan, E. Alata, V. Nicomette, and M. Kaâniche, "Bypassing iommu protection against i/o attacks," in *LADC*, 2016, pp. 145–150.

[45] B. Morgan, E. Alata, V. Nicomette, and M. Kaâniche, "Iommu protection against i/o attacks: a vulnerability and a proof of concept," *Journal of the Brazilian Computer Society*, vol. 24, no. 1, pp. 1–11, 2018.

[46] M. Mushtaq, A. Akram, M. K. Bhatti, M. Chaudhry, V. Lapotre, and G. Gogniat, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *HASP*, 2018, pp. 1–8.

[47] M. Mushtaq, J. Bricq, M. K. Bhatti, A. Akram, V. Lapotre, G. Gogniat, and P. Benoit, "Whisper: A tool for run-time detection of side-channel attacks," *IEEE Access*, vol. 8, pp. 83 871–83 900, 2020.

[48] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *MICRO*, 2017, pp. 354–366.

[49] Nvidia, "RDMA Aware Programming User Manual v1.7." [Online]. Available: https://docs.nvidia.com/networking/display/RDMAAwareProgrammingv17/RDMA+Aware+Networks+Programming+User+Manual

[50] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox: Practical cache attacks in javascript and their implications," in *ACM CCS*, 2015, pp. 1406–1418.

[51] C. J. Peglow and T. Eisenbarth, "Security analysis of hybrid intel cpu/fpga platforms using iommus against i/o attacks," 2020.

[52] H. Ragab, E. Barberis, H. Bos, and C. Giuffrida, "Rage against the machine clear: A systematic analysis of machine clears and their implications for transient execution attacks," in *USENIX Security Symposium*, 2021, pp. 1451–1468.

[53] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, "Crosstalk: Speculative data leaks across cores are real," in *IEEE S&P*, 2021, pp. 1852–1867.

[54] R. Recio, B. Metzler, P. Culley, J. Hilland, and D. Garcia, "A remote direct memory access protocol specification," Tech. Rep., 2007.

[55] Y. Ren, X. Wu, L. Zhang, Y. Wang, W. Zhang, Z. Wang, M. Hack, and S. Jiang, "irdma: Efficient use of rdma in distributed deep learning systems," in *HPCC*, 2017, pp. 231–238.

[56] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating Conflict-Based cache attacks with a practical Fully-Associative design," in *USENIX Security Symposium*, 2021, pp. 1379–1396.

[57] ——, "Tlb;dr: Enhancing tlb-based attacks with tlb desynchronized reverse engineering," in *USENIX Security Symposium*, 2022, pp. 989–1006.

[58] F. L. Sang, V. Nicomette, and Y. Deswarte, "I/o attacks in intel pc-based architectures and countermeasures," in *SysSec Workshop*, 2011, pp. 19–26.

[59] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard, "Keydrown: Eliminating software-based keystroke timing side-channel attacks," in *NDSS*, 2018.

[60] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher, and D. Gruss, "Zombieload: Cross-privilege-boundary data sampling," in *ACM CCS*, 2019, pp. 753–768.

[61] D. X. Song, D. Wagner, and X. Tian, "Timing analysis of keystrokes and timing attacks on SSH," in *USENIX Security Symposium*, 2001.

[62] P. Stewin and I. Bystrov, "Understanding dma malware," in *DIMVA*, 2012, pp. 21–41.

[63] P. Stuedi, A. Trivedi, J. Pfefferle, R. Stoica, B. Metzler, N. Ioannou, and I. Koltsidas, "Crail: A high-performance i/o architecture for distributed data processing." *IEEE Bulletin of the Technical Committee on Data Engineering*, vol. 40, no. 1, pp. 38–49, 2017.

[64] M. Tan, J. Wan, Z. Zhou, and Z. Li, "Invisible probe: Timing attacks with pcie congestion side-channel," in *IEEE S&P*, 2021, pp. 322–338.

[65] M. Taram, A. Venkat, and D. Tullsen, "Packet chasing: spying on network packets over a cache side-channel," in *ISCA*, 2020, pp. 721–734.

[66] F. Tian, Y. Zhang, W. Ye, C. Jin, Z. Wu, and Z.-L. Zhang, "Accelerating distributed deep learning using multi-path rdma in data center networks," in *SOSR*, 2021, pp. 88–100.

[67] T. Tiemann, Z. Weissman, T. Eisenbarth, and B. Sunar, "IOTLB-SC: An Accelerator-Independent Leakage Source in Modern Cloud Systems." *arXiv preprint arXiv:2202.11623*, 2022.

[68] A. Trivedi, B. Metzler, and P. Stuedi, "A case for rdma in clouds: turning supercomputer networking into commodity," in *APSys*, 2011, pp. 1–5.

[69] S.-Y. Tsai, M. Payer, and Y. Zhang, "Pythia: remote oracles for the masses," in *USENIX Security Symposium*, 2019, pp. 693–710.

[70] S. Van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida, "Ridl: Rogue in-flight data load," in *IEEE S&P*, 2019, pp. 88–105.

[71] Y. Wang, X. Meng, L. Zhang, and J. Tan, "C-hint: An effective and reliable cache management for rdma-accelerated key-value stores," in *SoCC*, 2014, pp. 1–13.

[72] Y. Wang, L. Zhang, J. Tan, M. Li, Y. Gao, X. Guerin, X. Meng, and S. Meng, "Hydradb: a resilient rdma-driven key-value middleware for in-memory cluster computing," in *SC*, 2015, pp. 1–11.

[73] X. Wei, R. Chen, H. Chen, and B. Zang, "Xstore: Fast rdma-based ordered key-value store using remote learned cache," *ACM Transactions on Storage*, vol. 17, no. 3, pp. 1–32, 2021.

[74] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *USENIX Security Symposium*, 2019, pp. 675–692.

[75] R. Wojtczuk *et al.*, "Subverting the xen hypervisor," *in Black Hat USA*, 2008.

[76] W. Wu, M. Crawford, and M. Bowden, "The performance analysis of linux networking–packet receiving," *Computer Communications*, vol. 30, no. 5, pp. 1044–1057, 2007.

[77] M. Xu, S. Liu, D. Yu, X. Cheng, S. Guo, and J. Yu, "Cloudchain: a cloud blockchain using shared memory consensus and rdma," *IEEE Transactions on Computers*, 2022.

[78] J. Xue, Y. Miao, C. Chen, M. Wu, L. Zhang, and L. Zhou, "Fast distributed deep learning over rdma," in *EuroSys*, 2019, pp. 1–14.

[79] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, l3 cache side-channel attack," in *USENIX Security Symposium*, 2014, pp. 719–732.

[80] K. Zhang and X. Wang, "Peeping tom in the neighborhood: Keystroke eavesdropping on multi-user systems," in *USENIX Security Symposium*, 2009, pp. 17–32.

[81] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *RAID*, 2016, pp. 118–140.

[82] Z. Zhu, S. Kim, Y. Rozhanski, Y. Hu, E. Witchel, and M. Silberstein, "Understanding the security of discrete gpus," in *GPGPU*, 2017, pp. 1–11.

Appendix A. Discussion of website fingerprinting attack

We discuss the performance of our website fingerprinting attack in terms of the noise and the use of huge pages.

1) Noise from other devices. To evaluate the website fingerprinting attack in a more practical setting, we perform the same experiment as described in Section 6.2 but this time we introduce noise on the IOTLB. In particular, we use an additional RNIC as a noise source, generating dummy DMA transactions that occupy certain IOTLB sets. We measure the classification accuracy by varying the amount of DMA transaction (i.e., the ratio of the occupied IOTLB sets). For

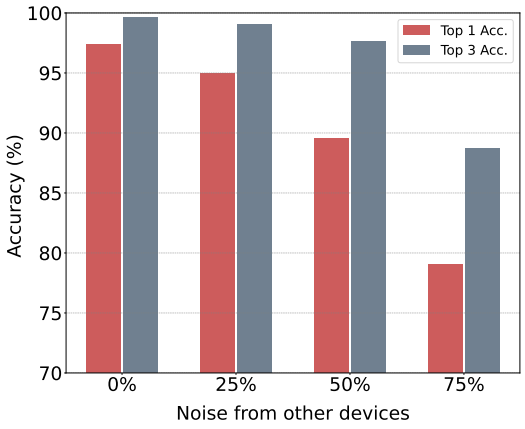


Figure 13: Classification accuracy according to the noise from other devices ($N = 100$).

example, to simulate 50% noise, the noise-generating RNIC device repeatedly accesses half of the IOTLB sets while the spy probes the IOTLB. Figure 13 shows the experimental results under the noise condition, which show a decrease in the classifier's accuracy as the noise increases. Specifically, the accuracy for Top 1 and Top 3 are decreased to 79% and 88.7%, respectively, when the dummy DMA transactions occupy 75% (24 sets) of the IOTLB.

2) Using huge pages. The website fingerprinting attack in Section 6.2 was evaluated on a web browser using GPU rendering on IOTLB for 4KB pages. In this section, we examine the performance of the attack in the context of GPU rendering on IOTLB for huge (2MB) pages. We use the same experimental setup presented in Section 6.2 for the experiments. The measurement results are presented in Table 7, which shows that the classification accuracy for Top 1 and Top 3 in the attack using huge pages decreased to 89.52% and 97.62%, respectively, compared to the attack using regular pages. We attribute this decrease in accuracy to the coarser-grained information contained in the IOTLB for huge pages.

TABLE 7: Classification accuracy for huge and regular pages ($N = 100$).

	Huge page (2MB)	Regular page (4KB)
Top 1 Acc.	89.52%	97.39%
Top 3 Acc.	97.62%	99.62%

Appendix B. Anomaly detector

We have developed an anomaly detector for detecting DEVIOUS attacks and evaluated its performance. Several well-established ML-based anomaly detection techniques have been developed for CPU-based side-channel attacks [7], [15], [46], [47], [81], all of which utilize hardware events for anomaly detection. Since our detector also utilizes hardware events for the IOMMU, detector can be easily incorporated into these existing techniques by slightly modifying previous implementations. Specifically, our anomaly detector is based on WHISPER [47], which employs an ensemble model of decision tree, random forest, and SVM for detecting attacks. Following the approach of [47], we constructed the detector by utilizing four IOMMU hardware events, which are detailed in Section 7. The classification models utilized by our detector were trained using specific hyper-parameters that are provided in Table 8.

TABLE 8: Hyper-parameters for the anomaly detector.

Model	Hyper-parameter	Value
Decision Tree	Max depth	2
	Splitter	best
	Max features	auto
Random Forest	N-Estimators	20
	Max depth	5
	Max features	sqrt
SVM	C	5
	Kernel	sigmoid
	degree	2

We evaluated the performance of our anomaly detector in terms of runtime overhead, accuracy, and F1-score under the same experimental setup presented in Section 5. With respect to runtime overhead, we measured the increase in CPU usage while the detector is running. The experimental result shows a 5% increase in CPU usage. Additionally, the detector exhibits an accuracy of approximately 98% for detecting DEVIOUS attack with a corresponding F1-score of 0.98, which indicates low false positive. The experimental results are detailed in Table 9.

TABLE 9: Experimental results of the anomaly detector.

	Precision	Recall	F1-Score	Accuracy
Normal	0.97	0.98	0.98	0.98
Attack	0.98	0.97	0.98	

Despite the promising performance of the anomaly detector, skilled attackers may still be able to evade it. One potential evasion technique involves the introduction of dummy DMA transactions to the IOMMU, which can potentially confuse the detector due to the increased noise. However, as discussed in Appendix A.1, this approach causing the noise may also disrupt the attack.

Appendix C. Mitigation against Attack 4

The application-level mitigation against Attack 4 could result in decreased performance of the web browser, particularly when loading web pages. We conducted an experiment to quantify the performance degradation. As the proposed method requires modification of web browsers, we decided to simulate the effects of dummy DMA transactions using an RNIC. We used the same experimental setup described in Section 6.2 for the evaluation. During the experiment, we measured the rendering time of web pages in Chrome as the RNIC generated DMA transactions accessing the

IOTLB sets, simulating the web browser generating dummy transactions. To measure the rendering time, we utilized the Chrome's performance panel [16], which enables profiling runtime performance of Chrome. Table 10 presents the results of our experiment, which shows the correlation between the amount of dummy transactions (i.e., the number of accessed IOTLB sets) and the measured rendering time. For instance, dummy DMA transactions accessing 16 sets of the IOTLB resulted in 1,811ms of rendering time, which is 2.8 times slower than the baseline without dummy transactions.

TABLE 10: The averaged rendering time of a web page (reddit.com) regarding the amount of dummy DMA transactions.

	Baseline	16 sets	32 sets
Rendering time (ms)	638	1,811	1,950

Appendix D. Examples of IOTLB traces for different websites

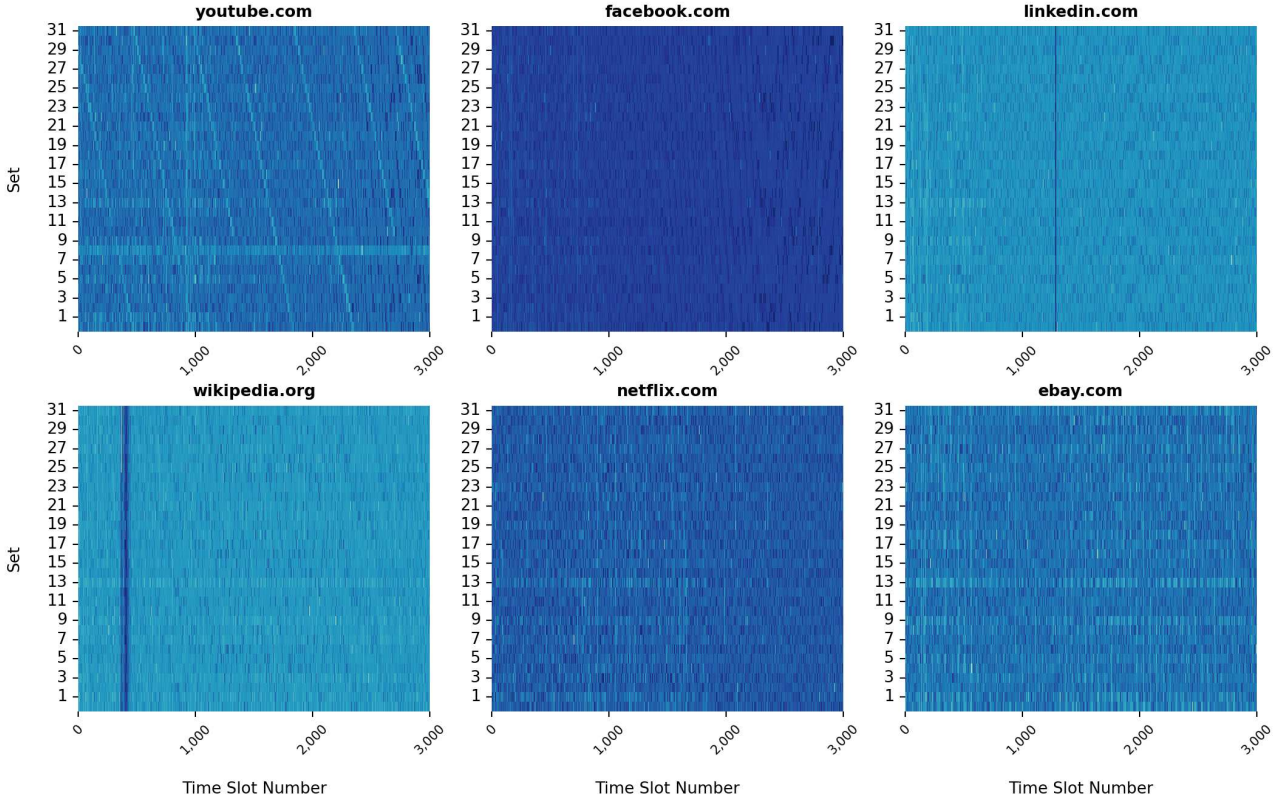


Figure 14: Examples of IOTLB traces for different websites. The brighter color indicates more IOTLB misses. The traces are obtained from 32 IOTLB sets for 7.7 seconds with the time slot interval of 2.5ms while each website is being loaded in Chrome.