NICraft: Malicious NIC Firmware-based Cache Side-channel Attack

Amit Choudhari[1], Shorya Kumar[1], and Christian Rossow[1]

CISPA Helmholtz Center for Information Security
amit.choudhari@cispa.de, shoryakumar411@gmail.com, rossow@cispa.de

Abstract. Being in the central position for network communication, Network Interface Cards (NICs) can intercept, modify, and generate network packets. But little is known about how NICs can launch attacks that go beyond simple traffic sniffing and manipulation.
In this work, we demonstrate how a malicious NIC can leverage a stock operating system to leak memory access patterns of user-space processes via a cache side-channel attack. Specifically, we uncover an exploitable gadget in the TCP/IP stack's IP reassembly process that enables a Prime+Probe cache side-channel attack. Our approach achieves fine-grained control, targeting individual cache sets. We validate its practicality by establishing a high-accuracy bi-directional covert channel between a user application and the NIC that bypasses firewalls, achieving transmission rates of 0.1 bits/sec upstream and 0.76 bits/sec downstream. In addition, we demonstrate its application in monitoring system activity by detecting periods of keystroke activity of 20 users across multiple sessions with a precision of over 96%, highlighting the potential privacy risks.

1 Introduction

Internal hardware devices are often overlooked as potential attack vectors, despite the inherent risks of malicious firmware. As Network Interface Cards (NICs), Graphics Processing Units (GPUs) and other peripheral devices typically prioritize performance and time to market over security, they are often found vulnerable to compromise. Numerous examples show how attackers can exploit these weaknesses: injecting malware to eavesdrop on network traffic [38], abusing firmware vulnerabilities [10], and exploiting buggy update tools [7]. Similarly, supply chain attacks or untrusted vendors can compromise firmware, as demonstrated by NIC backdoors that provide a shell on the GPU [37].

NICs are prime attack targets due to their Direct Memory Access (DMA) capabilities and role in mediating network traffic. Malicious firmware can enable eavesdropping [4] and traffic manipulation [11,19,25]. Furthermore, researchers have explored how malicious NICs can also directly exfiltrate sensitive system-level information [34]. However, countermeasures such as Input-Output Memory Management Unit (IOMMU) protections [3,13] constrain DMA access to specific memory regions, effectively defending against direct memory attacks.

Timing-based cache side-channel vulnerabilities present a unique threat as they operate without memory access authorization. IOMMU protections block memory attacks but not side channels. Traditionally, NICs interact with the CPU cache indirectly, transferring data to main memory using DMA, which is only fetched into the cache later by the driver. This latency limits direct control over Last-Level Cache (LLC) loads, making cache side-channel attacks challenging.

To accelerate packet processing, Intel introduced the Data Direct I/O (DDIO) hardware extension, enabling the NIC to directly transfer packets to the processor's LLC. This allows the NIC to directly manipulate and observe the LLC, simplifying cache side-channel attacks. Another extension, Remote Direct Memory Access (RDMA), allows remote machines to access pre-allocated memory regions directly. The combination of RDMA and DDIO facilitates network-based cache attacks [18]. However, these approaches rely on specialized hardware extensions that may be unavailable or disabled for security reasons, particularly in commodity systems. This raises a critical research question: *Can a malicious NIC orchestrate cache side-channel attacks without relying on these extensions?*

In this work, we introduce NICraft[1], a cache side-channel attack orchestrated by a malicious NIC, based on PRIME+PROBE (P+P) (Section 2.2) at cache set granularity. We exploit fragmented IP packet reassembly, a predictable mechanism in the Linux kernel's TCP/IP stack, to control memory access patterns and induce observable cache interference in (almost) arbitrary cache sets. NICraft leverages two kernel-level gadgets: one for priming, triggered upon receiving an IP fragment and another for probing, activated during final IP reassembly. The final packet's response delay reveals target address access via eviction in the target cache set. This enables NICraft to leak fine-grained system activity such as user keypresses, while bypassing traditional host- and network-based defenses.

Previous research has exploited predictable Operating System (OS) mechanisms, such as task scheduling in real-time systems [6] or file system synchronization [14], to mount timing-based side-channel attacks. However, these methods neither control cache access patterns nor involve cache interference. In contrast, NICraft uniquely exploits IP reassembly, a key operation in the TCP/IP stack, to construct cache interference gadgets and orchestrate side-channel attacks. By relying solely on standard NIC functionality, we show that a NIC can perform cache side-channel attacks without specialized hardware extensions. To the best of our knowledge, this is the first work to leverage predictable OS-level operations in the network stack for cache-based side-channel attacks. Thus, broadening the applicability of such NIC-based attacks to commodity systems.

Our experimental results demonstrate the impact of NICraft. We establish a high-accuracy covert channel between a user application and the NIC, achieving a data transmission rate of 0.1 bits/sec upstream and 0.76 bits/sec downstream. Furthermore, we use this side-channel capability to monitor system activity by detecting periods of keystroke activity across multiple users with a precision of over 96%. This highlights a previously unexplored threat model.

As part of NICraft, this work makes the following key contributions:

[1] `https://github.com/amit-choudhari/NICraft`

1. We identify an exploitable gadget in the Linux TCP/IP stack, enabling a malicious NIC firmware to execute a P+P attack at cache set granularity, without specialized hardware like DDIO or RDMA.
2. We develop signal amplification techniques (the *Aging* and *Domino effect*) and maximize cache set coverage through DHCP manipulation.
3. We demonstrate practical attacks bypassing on-system firewalls: a covert channel between the NIC and a user-space application, and an attack that detects system activity (such as keystroke activity) from the NIC.

2 Background

2.1 Network Interface Card (NIC)

A NIC facilitates communication between a computer and a network by transmitting data packets as signals over mediums like Ethernet or Wi-Fi. NICs can intercept, modify, and create packets while managing DMA operations, allowing direct data transfer between the NIC and main memory without CPU involvement. During initialization, the NIC driver allocates circular linked lists of buffers: RX ring buffers for incoming packets and TX ring buffers for outgoing packets. These buffers, shared between the NIC and the driver, match the NIC's Maximum Transmission Unit (MTU)–typically 1500 bytes for Ethernet. The driver also allocates descriptors specifying buffer sizes and physical addresses, enabling the NIC to perform DMA transfers. At any point, DMA is restricted to the RX/TX rings, i.e., a NIC cannot access memory globally.

To transmit data, an application creates a socket buffer (skb). The NIC driver retrieves the data from the skb and prepares it for transmission, fragmenting larger message into MTU-compliant packets. These packets are placed in the TX ring for processing and will be transmitted over the network eventually.

To receive data, the NIC processes incoming packets and transfers them to main memory via DMA. It then interrupts the driver, which maps RX buffers to socket buffers (skb) for TCP/IP stack processing. This avoids an additional memory copy. If an IP packet is fragmented during transmission, its fragments are validated within the TCP/IP stack of the OS, including header checksum verification. Once the final IP fragment arrives, the kernel reassembles them into the original packet respecting the offsets to ensure proper order.

2.2 Cache Organization and Side-Channel Attacks

Cache Hierarchy Modern processors use hierarchical caches (Level 1 (L1), Level 2 (L2), and LLC) to reduce memory latency. These hierarchies often follow inclusivity, ensuring that data in lower-level caches (L1/L2) also resides in the LLC, shared across cores. While this simplifies cache management, it also makes the LLC vulnerable to side-channel attacks due to its shared and predictable structure. The LLC is structured into sets and slices for efficient storage. Sets are logical groups of cache lines, with memory addresses mapped to sets based on specific bits. Slices are physical divisions enabling parallel access to different LLC parts, improving efficiency and reducing contention.

Replacement Policies Cache replacement policies determine which cache line to evict when inserting new data into a full cache set. The Least Recently Used (LRU) policies, most commonly found in LLC, evict the longest-unused cache line, leveraging temporal locality for efficiency but adding hardware complexity. Pseudo-LRU policies mitigate this complexity by approximating LRU behavior with lower overhead and comparable performance.

Cache Side Channel Attacks Side-channel attacks monitor state changes in shared resources like the LLC to infer a victim's activity. PRIME+PROBE (P+P) is a popular timing side-channel technique to uncover memory access patterns. In P+P, the attacker primes a specific cache set by initializing it to a known state. After the victim executes operations that may evict attacker's cache lines, the attacker probes the same set, measuring access times. Longer access times indicate cache miss, revealing the victim's memory access patterns.

3 Threat Model

Our threat model targets standard computers, including server and non-server CPUs, that use NICs for network communication. We consider a scenario where the attacker controls the NIC firmware to monitor and inject packets, to control Interrupt Requests (IRQs) and thus influence kernel packet processing.

Using these capabilities, the attacker orchestrates a cache side-channel attack through the malicious NIC. The NIC crafts packets to exploit predictable TCP/IP stack operations, causing timing variations and allowing the attacker to infer memory access patterns on specific cache sets with precision. The attacker knows (or can deduce) the cache sets used by the victim process [29]. Strategically, the attacker places packets in the NIC's RX queue to induce interference in targeted cache sets. The attacker's firmware uses a reliable timing source or an equivalent mechanism, such as counter threads.

The model assumes an unmodified Linux OS and does not require DDIO or RDMA extensions, enabling the attack on commodity systems. No other microarchitectural behaviors such as speculative execution are exploited, relying instead on standard NIC capabilities and TCP/IP stack cache side-channel leakage.

4 Methodology

This section explains NICraft, beginning with an overview and an analysis of exploitable gadgets in the TCP/IP stack. It then addresses practical challenges in achieving fine-grained observations of single cache line evictions.

4.1 Attack Overview

Figure 1 illustrates the general idea of NICraft in which the attacker uses malicious NIC firmware to orchestrate a cache side-channel attack, leaking memory

```
1  static int ip_frag_queue(struct ipq
       *qp, struct sk_buff *skb)
2  {
3    err = pskb_trim_rcsum(skb, end -
         offset);
4    if (LAST_FRAGMENT)) {
5      ...
6      err = ip_frag_reasm(qp, skb,
           prev_tail, dev);
7      ...
8      return err;
9    }
10   ...
11 }
```

Listing 1.1: IP fragmentation method invoked on receiving each fragment. Blue: priming checksum function. Red: final fragment's reassembly function.

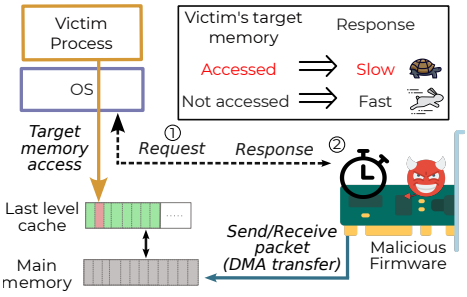


Fig. 1: Attack overview: Black arrow shows TX-RX flow, green blocks represent LLC ring buffer data, and the red block indicates secret-dependent memory access evicting packet data.

access patterns of a victim process on the host CPU. NICraft operates in two steps: (1) The NIC sends a crafted request to the host, triggering a side-channel gadget in the TCP/IP stack to process the request and generate a response. (2) The NIC measures the response delay to infer the victim's access to the target memory. These crafted packets leverage standard TCP/IP mechanisms without requiring user-space services. For example, a TCP SYN sent to an unbound TCP port triggers a TCP RST—the timing of which depends on the cache state. During packet processing by the kernel and driver, headers and data are loaded into the cache. The NIC firmware maps packet data to the same cache set as the victim's memory to observe access patterns. By managing Interrupt Requests (IRQs), the NIC aligns the timing of packet processing with the victim's memory access, inducing observable timing variations due to cache collisions.

The NIC extracts access patterns from user-space using the P+P technique, based on the observation that the TCP/IP stack (i.e., the kernel) accesses some packets *twice*. For priming, the NIC enqueues packets to occupy the target cache set X. Next, the NIC either triggers the victim application or waits for the victim to access the target memory within the same cache set X. When the victim accesses this memory, at least one of the NIC's packet cache lines is evicted based on the cache replacement policy. For probing, the NIC sends a packet to *reload* previously cached packets. A processing delay indicates victim evictions and is incorporated into the packet response time. The NIC measures delays with an internal timer, where longer delays indicate evictions in cache set X.

Leveraging IP Reassembly A critical insight of NICraft is that it can provoke the OS kernel to access the same packet twice. Typically, a single memory access suffices to process a packet. The NIC transfers a packet into main memory via DMA and raises a hardware interrupt to notify the driver. The driver, followed by the network stack, consumes these packets in ring buffer order. During

```
1  bool skb_try_coalesce(struct
       sk_buff *to, struct sk_buff *
       from, ...)
2  {
3    if (len <= skb_tailroom(to)) {
4      BUG_ON(skb_copy_bits(from, 0,
5        skb_put(to, len), len));
6    }
7  }
```

Listing 1.2: Function to coalesce fragmented data into the tail of the first fragment, red higlights data copying function.

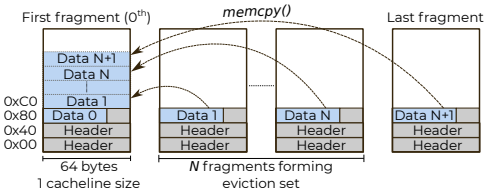


Fig. 2: *reassembly_load* gadget: Loads data block of N fragments into cache to copy them to the tail of first (0th).

processing, the packet is loaded into the cache. Afterward, the memory is freed and not accessed again. Without additional measures, the NIC cannot trigger a second access to the same memory location.

However, we identified two gadgets in the TCP/IP stack that load packet data into the cache at *two distinct times*. The Internet Protocol (IP) fragments packets into smaller pieces if their size exceeds the MTU. These fragments are accessed again when the kernel reassembles the original IP packet before passing it to higher layers. NICraft leverages IP reassembly for P+P as follows:

P+P Priming: When a host receives a fragmented packet, it maps the packet's fields into the *sk_buff* struct. As shown in Listing 1.1 (line 3), the host verifies the checksum of the IP header and stores the fragments in a queue until all fragments of an IP packet have been received. If the header and associated data share a cache line, they are together loaded into the cache. This partial processing creates the *checksum_load* gadget that we leverage for P+P priming.

P+P Probing: To verify if the victim's memory access evicted fragments from the cache, the NIC triggers reaccess of the primed cache lines. As shown in line 6 in Listing 1.1, upon receiving the final fragment, the host coalesces all fragments before passing the complete packet to the transport layer. As shown in Listing 1.2, fragments smaller than the RX buffer are coalesced by copying their data into the tail of the first fragment before being freed (overcoming side-effects of *fast_string* operation Appendix A). This copying reaccesses the primed fragments, creating the *reassembly_load* gadget for P+P probing.

Controlling a Cache Set NICraft creates an eviction set for an N-way cache. The NIC uses DMA-based physical addresses to map packets to cache sets and slices [28]. As shown in Figure 2, the first fragment (*Data 0*) is not reloaded into the cache, making it unusable for the attack. Likewise, the last fragment, which triggers *reassembly_load* during the probe phase, is not suitable for priming. To address these exceptions, the NIC creates $N + 2$ fragments, excluding the first and last fragments from the target cache set. The middle N fragments are accessed twice: first during checksum validation (*checksum_load*) and again during reassembly when the final fragment completes the IP packet (*reassembly_load*).

The host processes packets sequentially, forcing the NIC to copy them in RX ring order. To align N fragments in RX buffers and cache them in the same set, the NIC interleaves malformed packets (we refer to as *dummy_pkts*) between fragments. The driver drops these *dummy_pkts*, reducing work and noise.

However, the NIC faces an additional challenge to target cache sets beyond the memory ranges allocated for RX buffers. The MTU-constrained buffer size initially limits the NIC to a couple hundred cache sets. Varying the MTU size enables access to additional unique cache sets. We broaden NICraft's scope to target more cache sets by dynamically adjusting the MTU without OS-level modifications. We leverage the DHCP renewal process with Option 26 to dynamically modify the MTU size. Adjusting the MTU size triggers a corresponding resizing of the ring buffers, enabling the NIC to target a wider range of cache sets.

Furthermore, we scan the targetable eviction sets for MTU sizes between 1000B and 9216B. Note that smaller MTUs that could not accommodate $N+2$ fragments' data were excluded, as that is the prerequisite for NICraft. The scan identifies 78% of all cache sets (6400 out of 8192) on the Intel Skylake CPU, as shown in Figure 7 on page 11. These results highlight NICraft's scalability, as MTU adjustments efficiently target most cache sets.

Synchronization and Noise Reduction To leak the victim's access patterns, NICraft requires the NIC to synchronize with minimal noise. The attacker faces three key synchronization challenges: (1) Minimize noise during the priming phase. (2) Control the interval between priming and probing, with minimum noise. (3) Trigger victim to access memory associated with sensitive operations.

Minimizing noise during priming: Sending $N+2$ fragments and hundreds of *dummy_pkts* often triggers multiple IRQs due to network processing constraints. Multiple Interrupt Service Routines (ISRs) and delayed packet consumption disrupt the priming phase by polluting the cache. NICraft mitigates this by ensuring all crafted packets are processed in a single IRQ. The NIC disables interrupts until all packets are copied to main memory, forming a 'tape' of packets. The NIC then raises a single interrupt to ensure all packets are consumed sequentially.

P+P timing: The attacker controls the interval between prime (*checksum_load*) and probe (*reassembly_load*) phases by introducing delay before the last fragment. Delaying the last fragment via a new IRQ pollutes the cache. To mitigate this, NICraft inserts *dummy_pkts* into the RX queue before the last fragment and raises a single IRQ afterward (see Figure 3). The driver processes these packets sequentially, delaying the probe phase. The *dummy_pkts* packets, discarded early, cause minimal cache pollution while delaying the probe.

Triggering the victim: In NICraft, the attacker triggers the victim to access target data by sending crafted TCP/IP packets, timing attacks with legitimate requests like TLS handshakes, or aligning with periodic tasks. For instance, logging into a service may load sensitive data into the cache during encrypted session handling. Alternatively, by repeatedly performing P+P, the attacker monitors non-periodic activities and detects specific events. This approach allows the attacker to infer system activity, user activity, or specific access patterns.

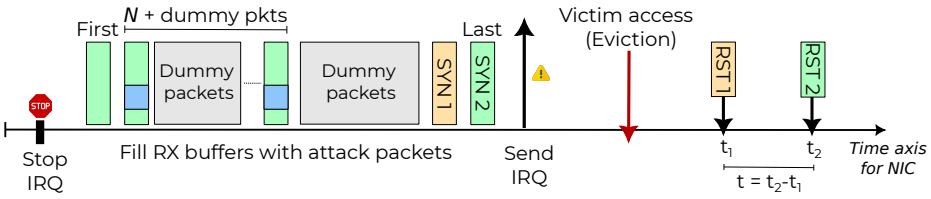


Fig. 3: The NIC inserts N IP fragments with data colliding in the target set. In the prime phase, TCP/IP stack computes checksum of these fragments. In the probe phase, a final fragment ("SYN2") triggers the reassembly. The interval t between two RSTs reflects reassembly speed, indicating if the victim evicted a fragment.

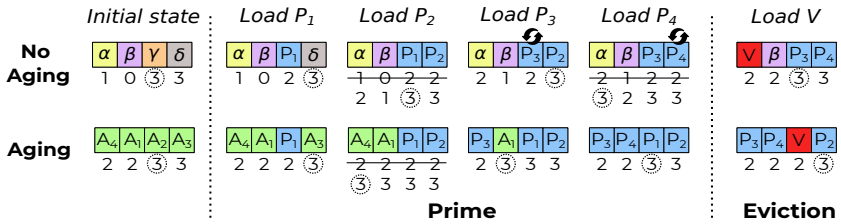


Fig. 4: Impact of *Aging* on cache priming: Top shows *without Aging*, bottom shows *with Aging*, ensuring victim evicts attacker-controlled fragments. Dotted circle on age: next eviction candidate, recycle arrows: *Self eviction*

Time Measurement The host processes all packets (TCP fragments and *dummy_pkts*) during a single ISR, and the NIC receives an acknowledgment (RST) only after the last TCP fragment is processed. This time frame includes scheduling and processing hundreds of packets, introducing noise and weakening the eviction signal. As shown in Figure 3, NICraft extracts reassembly time by adding a new SYN segment (*SYN1*) just before *SYN2* in the RX ring buffer, generating *RST1* and *RST2*. As packets are consumed sequentially, *RST2* timing (t_2) depends on the cache eviction event. The NIC measures the timing variation (t) between these two RSTs to infer the victim's access patterns accurately.

4.2 Towards Cache-set Granularity Attacks

Building on NICraft's fundamentals, we explore two challenges of measuring single cache line evictions from a NIC in practical settings. First, the priming fragments must fully occupy the target cache set. If the cache set is not fully dominated by the attacker's data, the victim may fail to evict attacker's data, compromising the accuracy of the attack. Second, amplifying the weak signal from a single cache line miss is essential, as it may go undetected by the NIC.

Cache Set Aging Loading a new cache line into a fully occupied N-way cache set requires evicting an existing line. This eviction decision is governed by the

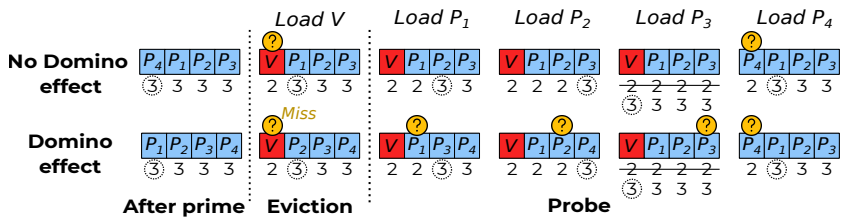


Fig. 5: Eviction comparison: Top (*without Domino effect*): Only P_4 is evicted; Bottom (*with Domino effect*): Sequentially evicting all fragments during probing

cache replacement policy, which tracks the age of each cache line. For example, Intel Skylake processor's LLC follows the Quad-Age LRU (QRLU) replacement policy [2], where each cache line is assigned two bit age (ranging from 0 to 3). New cache lines are assigned an age of 2, which decrements with each access until reaching 0. When no free slots are available, the oldest cache line (age 3) is evicted. If no line has an age of 3, all ages increment until one reaches age 3.

Variability in initial cache line ages and complex replacement policies, such as adaptive and QLRU, makes it challenging to prime the cache set with a single access. This unreliability increases the risk of the attacker missing the victim's eviction. The top half of Figure 4 (*No Aging*) shows a 4-way cache where initial line ages are 1 (α), 0 (β), 3 (γ), and 3 (δ). In this case, γ—being the leftmost cache line with age 3—is the next eviction candidate. The attacker attempts to fill the cache set (from left to right) by sequentially loading P_1, P_2, P_3, and P_4 (four fragments of a packet P). However, due to *Self eviction* (new cache load (P_3) evicts earlier fragment (P_1)), α and β remain in cache. This prevents the set from being fully primed by P and causes the attacker to miss the eviction.

A more practical approach, referred to as *Aging* (Appendix B), creates an eviction set larger than N to increment cache line ages to 2 or 3. The bottom half of Figure 4 (*Aging* case) shows a cache set after *Aging*, with ages set to 2 (A_4), 2 (A_3), 3 (A_2), and 3 (A_1). A_2, the leftmost cache line with age 3, is replaced by P_1. This process continues until P_4 fills the entire cache set. In the *No Aging* case, α and β have lower initial ages, making them harder to evict. In contrast, $A_{1\text{-}4}$ have higher ages, preventing *Self eviction*. Age disparity primarily causes *Self eviction*. Cache lines aged in pattern (2^*3^+) (i.e., consist of zero or more lines with age 2, followed by at least one line with age 3) act as an empty cache set, enabling the next N cache misses to fully occupy it.

Having assessed the impact of the preparatory step, we examine the process of reaching it. Due to the likelihood of *Self eviction*, filling an N-way cache set with an eviction set of exactly N cache lines is difficult. However, the LRU cache replacement policy mitigates this issue as the eviction set grows, which gradually reduces the disparity in the ages of cache lines. Increasing the eviction set size increments the ages of lower-aged cache lines, resulting in uniform aging.

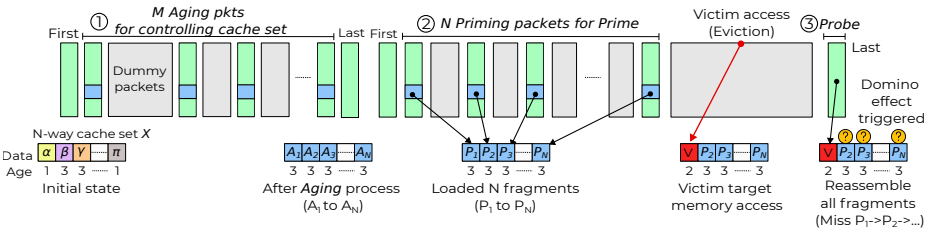


Fig. 6: End-to-end attack: (1) send M *aging* packets to prepare the cache set X, (2) send N priming packets to prime the cache set and enable the victim to evict one cache line P_n, and (3) send Last packet to trigger *Domino effect*

Domino Effect The NIC, connected via the PCIe bus, operates at a lower frequency than the host CPU–e.g., 1 GHz for a Netronome Agilio CX SmartNIC versus 3.4 GHz for an Intel Skylake CPU. A single LLC miss on the host incurs 250 cycles of latency, but the NIC perceives this delay indirectly via TX events. The periodic polling of the TX ring by the NIC adds further delays. These factors make it challenging for the NIC to detect individual cache line misses.

In NICraft, we introduce a novel technique, *Domino effect*, which amplifies the eviction signal by leveraging *Aging* as a preparatory step. This enhances timing differences from cache evictions, allowing the NIC to distinguish cache hits from misses. We leverage the QRLU replacement policy to amplify a single cache miss into multiple misses, making them detectable by the NIC. The key idea is to evict the next fragment to be probed, triggering a cascading chain of cache misses during IP reassembly. Figure 5 compares the impact of two scenarios on a 4-way cache: one without (top) and one with *Domino effect* (bottom), respectively. P_1–P_4 represent the four fragments of a packet P used for priming.

Without the *Domino effect* (top), after the priming, let P_4 be the next eviction candidate as it has age 3 and is on the leftmost side. When the victim accesses its sensitive data, it replaces this cache line (P_4) with its own data (V), assigning the new line an insertion age of 2. During the probe phase, all the fragments are accessed sequentially, with only a single cache miss occurring. Specifically, P_1, P_2, and P_3 are hits, while P_4 is a miss.

In contrast, triggering the *Domino effect* (bottom), forces the QLRU policy to amplify a single eviction's impact. Unlike the previous case, after the prime phase, P_1 becomes the next eviction candidate, and when the victim accesses its data (V), it evicts P_1 and assigns its cache line an insertion age of 2. In probe phase, sequentially accessing the fragments triggers a cascading chain of cache misses—the 'domino' effect. Loading P_1 evicts P_2, loading P_2 evicts P_3, and so on. This *Domino effect* occurs because priming arranges packets in sequential eviction order. Furthermore, both priming and probing phases access the cache lines in the same order. A single victim access disrupts the LRU pattern. In this 4-way cache, the *Domino effect* amplifies a single cache line miss into four.

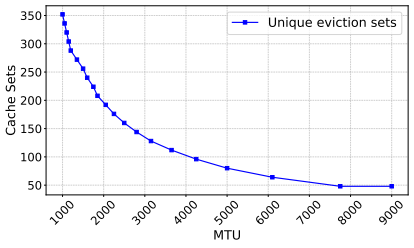


Fig. 7: Targetable cache sets for corresponding MTUs

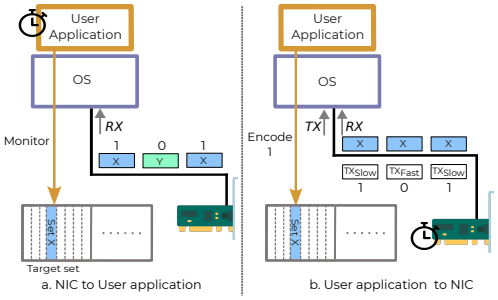


Fig. 8: Covert channel attack: NIC to App (left) and App to NIC (right).

End-to-End Attack Design As shown in Figure 6, NICraft begins with the NIC sending *Aging* packets, A, to prime the cache. These packets mimic fragmented TCP segments, with the number of fragments (M) exceeding the cache's associativity (N), i.e., $M > N$. The *Aging* phase overfills the cache set, ensuring that it is fully primed for eviction in the subsequent phase. The NIC ignores responses from *Aging* packets. Next, the attacker initiates the PRIME+PROBE attack by sending N priming fragments, P. These P packets target the same cache set used during the aging phase. The NIC organizes A and P packets in its ring buffer: the first M packets for *Aging* and the next N for priming. By carefully crafting both A and P to match an eviction set of size $M + N$, the attacker triggers the *Domino effect*, amplifying a single cache miss into multiple observable misses, enabling NICraft to reliably detect the side-channel signal.

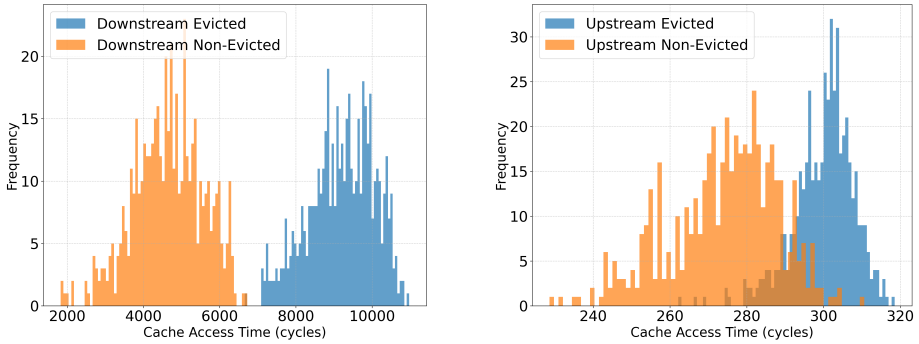
5 Attack Use Cases

5.1 Experimental Setup

We evaluate NICraft on an Intel Skylake i7-6700 CPU with stock Linux 6.2.16, using a Netronome Agilio CX 2x10GbE NIC with programmable firmware to install malicious code. Notably, attackers are not limited to reprogrammable NICs; malicious code can be introduced via proprietary firmware updates or compromises. The firmware handles packet modification, DMA transfers, IRQ control, and the timer. For simplicity, a remote server crafts fragmented TCP SYN packets and transmits them to the victim server. The NIC intercepts, modifies, and utilizes these packets to perform the attack. Alternatively, the NIC could generate these packets directly, reducing external dependencies.

5.2 Bi-Directional Covert Channel

While a malicious NIC could communicate through hidden packet modifications or through collaborating user-space processes opening listening ports, such channels can be readily blocked by firewalls or network policies, and they fail when



(a) Downstream Eviction Histogram (b) Upstream Eviction Histogram

Fig. 9: Cache Timing Analysis of the Covert Channel.

applications have no network access. In contrast, cache-based covert channels avoid network visibility altogether, enabling stealthy communication through microarchitectural interference. NICraft establishes a bidirectional covert channel between the NIC and a user-space application, focusing on reliability over raw throughput. By exploiting cache interference rather than IP traffic, the NIC and a user-space application exchange data even under strict network restrictions.

The NIC and the user-space app communicate over the N-way LLC by priming and probing two agreed-upon cache sets: U for upstream (app to NIC) and D for downstream (NIC to app). A 1 is encoded as eviction, and a 0 as no eviction.

For downstream transfers (Figure 8 left), the app monitors cache set D in three steps. It first primes D by loading one or more cache lines. Next, to send a 1, the NIC fragments a TCP SYN into $N+2$ IP packets targeting D, evicting the app's data. To send a 0, packets are crafted without evicting D. Finally, the app probes D; data absence indicates 1, presence indicates 0.

For upstream transfers (Figure 8 right), the NIC primes U using two fragmented TCP SYNs, S_1 and S_2. Fragments in S_1 perform *Aging* on U, while $N+2$ IP fragments in S_2 prime U. Next, the app encodes 1 by evicting cache lines from U, or skips eviction to encode 0. Finally, the NIC exploits *reassembly_load* to trigger the *Domino effect*, delaying fragment reassembly and the TCP RST response. A delayed RST indicates 1, and a timely RST indicates 0.

Evaluation We evaluate the covert channel's reliability, efficiency, and timing characteristics using a 1000-bit stream for both communication directions. Each bit is transmitted 15 times for downstream and 75 times for upstream, with each transmission lasting 0.1 ms and 0.3 ms, respectively. The counts (15 and 75) were heuristically chosen to balance throughput and error rate.

We measure the covert channel's reliability using the Bit Error Rate (BER). Downstream achieves a BER below 0.9%, demonstrating robustness. In contrast, upstream has higher BER of 10% due to the NIC's clock challenges. These dif-

ferences are also reflected in the timing histograms in Figures 9a and 9b. For the downstream channel, Figure 9a reveals a clear separation between evicted and non-evicted cache access times, with negligible overlap. The upstream channel's timing overlap, as shown in Figure 9b, is due to the noisy channel, contributing to higher BER. Error correction codes can mitigate this higher BER.

We evaluate the channel efficiency by measuring the throughput. Upstream achieves about 0.1 bit/s, while downstream maintains a consistent 0.76 bit/s. The upstream transmission time is slower due to two factors. First, higher noise in the channel leads to retransmission. The NIC's slower clock limits precise time variation measurements, while polling mechanisms delay RST packet notifications. Furthermore, indirect processing adds latency via the NIC, main memory, and cache. Second, the NIC has to search for the ring buffers in the target cache set; failure to find the target eviction set leads to flushing the buffers.

5.3 Run-time Detection of System Activity

Systems exhibit privacy leaking activities such as typing, active processes, peripheral usage, or cryptographic operations (e.g., YubiKey encryption). A malicious NIC could infer *some* of these activities by monitoring network traffic. However, this approach fails for local (i.e., non-networked) activities, or in scenarios like offline systems (e.g., military deployments) and systems with multiple interfaces (e.g., Wi-Fi vs. LAN vs. 5G). In such cases, a NIC can leverage our P+P attack to infer system activity without relying on network traffic.

To demonstrate this using NICraft, we use detection of periods of keystroke activity as a concrete example. Typing activity triggers kernel routines, leaving measurable traces in the LLC. By leveraging P+P, the NIC detects these traces and infers keystroke activity timing without direct access to the keyboard or OS input subsystem.

Our experiment targets the popular *libinput* library, which manages input events and remains active unless the compositor restarts. The attack detects execution of the *libinput_event_get_keyboard_event* function. First, the NIC primes the target cache set aligned to the function's entry point. Next, it waits for a keypress to trigger the target function. Finally, it probes the cache set. When no keypress occurs, the packet reassembly completes quickly as all fragments are cached. A keypress evicts a fragment, potentially triggering a *Domino effect*, delaying packet reassembly and the corresponding RST.

To distinguish keypress events from idle states, the attacker collects response time samples t_{kp} and t_{idle}, and calibrates the threshold t_{thr} as their midpoint. In real-time, the attacker collects W response time samples using a sliding window. The samples above the 80th percentile are averaged, and the mean is compared to t_{thr}. If it exceeds t_{thr}, a keypress event is detected. Both W and the 80th percentile threshold were heuristically chosen to balance accuracy and noise.

Evaluation We evaluate NICraft using a widely adopted typing activity dataset [15] containing data from 20 subjects typing free and transcribed text. We group

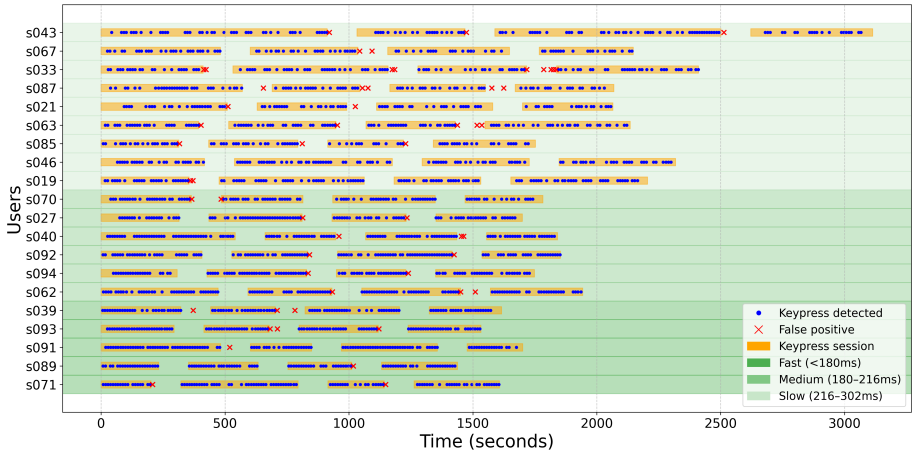


Fig. 10: Illustrates the keystroke experiment for 20 users. Valid detections (blue dots), actual typing activity (yellow highlights) and FP (red crosses).

Typing Speed	TP	FP	FN	TN	Precision	Recall	Specificity
Fast	0.78	0.05	0.22	0.94	0.981	0.780	0.947
Medium	0.70	0.06	0.29	0.93	0.979	0.707	0.938
Slow	0.52	0.09	0.48	0.90	0.967	0.518	0.906

Table 1: Summarizes the key performance metrics for different typing speeds.

subjects by typing speed based on their keystroke intervals: fast typists average 127-177 ms (σ: 108-115 ms), while medium typists range from 180-216 ms (σ: 127-158 ms) and slow typists range from 216-302 ms (σ: 144-168 ms). We introduce idle periods between the typing sessions to evaluate the attack's specificity. Figure 10 shows a visual representation of the experiment.

Table 1 summarizes performance metrics, including True Positive (TP), False Positive (FP), precision, recall, and specificity for different typing speeds. The attack achieves over 96% detection precision consistently. Most FPs were attributed to windowing effects, such as overlaps with idle phases or delayed detections during transitions. For example, in the *Fast* scenario, 6 out of 10 FPs occurred immediately after activity. Longer pauses during typing–more common among slower typists–contributed to increased False Negative (FN), lowering recall. To ensure high detection confidence, NICraft's design intentionally prioritized reducing FP over FN, striking a balance between sensitivity and reliability.

6 Related Work

DMA, and NIC-Based Attacks NICs use DMA for efficient data transfer. Despite protections like the IOMMU, attacks such as Thunderclap [26] and others [34,27] show how misconfigurations or bypasses enable malicious peripherals

to inject code or overwrite memory. Defense techniques such as secure architectures [41] and firmware verification [20] exist but suffer from high overhead.

Previous research has shown malicious NIC firmware can eavesdrop on traffic or stage backdoors [37,38], bypass IOMMU to perform unauthorized access [10,9]. Growing firmware complexity [4] compounds the attack surface, complicating secure firmware development. IOMMU restricts unauthorized DMA, but it cannot prevent timing-based attacks like cache side-channels. Unlike prior malicious NIC attacks which aim for active memory compromise or traffic interception, NICraft passively leaks system activity via cache interference, exploiting predictable TCP/IP stack behaviors to bypass IOMMU protections without direct memory access.

Microarchitectural attacks Early works [30,24] demonstrated P+P's effectiveness in cryptographic key extraction and cross-VM attacks, later extending to browser-based attacks [29,21], speculative execution [16], and prefetcher-based side channels [31].

A critical enabler for P+P is constructing eviction sets—groups of memory addresses mapping to the same cache set [40,28]. Our approach uses kernel-allocated NIC ring buffers to build these sets, following the method by Maurice et al. [28]. Tools such as CacheQuery [39], nanoBench [2], and other studies [5] identify cache replacement policies. Building on prior research, NICraft introduces novel kernel gadgets for P+P, innovative signal-amplification techniques (*Domino effect*, *Aging*) and dynamic MTU variation (via DHCP Option 26) to expand the range of targeted cache sets.

Network side-channel attacks Prior attacks exploit shared resources or timing variations caused by network interfaces and traffic processing to infer sensitive information. NetCAT [18] leverages Intel's DDIO to establish covert channels between remote clients and sandboxed processes or isolated clients on a victim machine. By exploiting the shared cache state between the NIC and CPU enabled by DDIO, NetCAT can perform attacks like keystroke timing inference over an SSH connection. However, the attack depends entirely on DDIO, which is absent in consumer-grade NICs and can be disabled, limiting its practicality. In contrast, NICraft operates without relying on DDIO, broadening the attack surface to commodity systems.

Unlike NICraft, Packet Chasing [35] assumes an attacker on the host CPU. The attack primarily leaks downstream data. By mapping cache sets to NIC ring descriptors, it reveals details of inbound traffic, such as packet size and timing. The host-based attacker can directly observe cache interference, easing downstream data leakage without relying on DDIO. The attack depends on stable descriptor reuse and becomes noisier with frequent changes in driver configurations, ring buffer allocations, and particularly when DDIO is disabled. NICraft differs by mounting the attack from the NIC itself, without requiring host-side software execution.

Other works, such as NetSpectre [32] exploit speculative execution vulnerabilities to leak data remotely but requires hard-to-identify Spectre gadgets. NetHammer [22], on the other hand, targets DRAM Rowhammer vulnerabilities by crafting network packets, inducing bit flips in memory. NICraft instead exploits predictable OS-level behaviors to induce microarchitectural cache contention without speculative execution or memory corruption.

NICraft broadens network side-channel threats by showing that a malicious NIC can leak host activity purely through cache interference, without relying on DDIO, speculative execution, or privileged code.

7 Discussion and Mitigation

Limitations and Improvements Our current method focuses on detecting local (non-networked) activities—a stricter scenario often overlooked by traffic-based inference (e.g., in offline or multi-interface environments). By not relying on any network-layer cues, we demonstrate the inherent power of a malicious NIC to infer system activities solely via side-channels. Integrating network knowledge (e.g., synchronizing with packet flows) would make such attacks more potent.

The attack is limited to cache sets at fixed offsets in ring buffers, hindering optimal eviction set construction. It targets one cache set at a time because identifying multiple sets within the current RX buffers is challenging. Overcoming this limitation might enable simultaneous attacks on multiple cache sets; e.g., AES T-tables require monitoring 16 cache sets simultaneously.

Our implementation is fine-tuned for an Intel Skylake platform, overcoming microarchitectural behaviors such as fast-string optimizations and leveraging the observed QLRU-based cache replacement policy. Broader applicability to other platforms was not evaluated and may require calibration.

Generalizability Although NICraft was evaluated on a specific platform, its core techniques remain broadly applicable within x86 systems. The Domino and Aging strategies exploit fundamental cache behaviors—specifically, that recently accessed cache lines are prioritized over older ones. Such aging effects are common across x86 processors, which employ QLRU approximations in LLC replacement policies [12], as confirmed by empirical studies [2,1].

While the overall attack protocol remains unchanged, adapting NICraft to other environments would require two categories of adjustments. First, porting to different operating systems would involve selecting alternative kernel-level gadgets that exhibit predictable cache access patterns, depending on how IP reassembly or similar functionality is implemented. Second, adapting to different CPU architectures would require recalibrating system-specific parameters such as cache set mappings, fragment sizing relative to cache line size, and buffer layouts. These parameters influence eviction timing, Aging and Domino amplification, and cache interference visibility. We leave exploring these adaptations to future work.

Mitigation To mitigate our attack, one approach is to statically or dynamically partition the LLC [23]. For example, Intel CAT [12] partitions cache access by *Class of Service* per core. Isolating the kernel from user caches can effectively prevent memory access pattern leaks across privilege levels. However, unless we also isolate different kernel threads, access patterns *within* the kernel would still leak. Hardware-based PMU performance counters may detect unusual cache activity dynamically and terminate the attack at runtime [8]. Nevertheless, such detection mechanisms are vulnerable to camouflage techniques [17], where the attacker mimics legitimate activity to evade detection.

NIC-driven anomalies, such as excessive small fragmented packets or unusual offsets, also reveal malicious behavior. In the driver, especially when unjustified by normal conditions, provides another detection mechanism. Similarly, techniques like CloudRadar [42], which detect side-channel attacks by correlating multiple cache anomalies, demonstrate the feasibility of anomaly-based approaches to flag malicious NIC-driven fragmentation patterns.

Beyond detection, proactive software defenses can further disrupt the attacker's ability to mount a reliable side-channel. Randomizing network buffer allocations and introducing jitter in fragment handling disrupt cache set predictability, degrading the attack's success rate with minimal performance overhead. It is non-trivial to avoid priming since the kernel must access packet headers (e.g., sequence numbers). Similarly, probing remains unavoidable because, even if data copying is deferred during reassembly, the data must eventually be combined before delivery to userspace. This issue likely extends to other kernel-based TCP/IP stacks, as it arises from fundamental operations inherent to their design. For fragmented packets, we propose a software-based mitigation to flush the header cache line after processing the header. This ensures that any data sharing the same cache line as the header is accessed directly from main memory.

8 Conclusion

In this work, we introduced NICraft, a malicious NIC-orchestrated P+P cache side-channel attack that exploits IP reassembly in the kernel's TCP/IP stack. Relying solely on standard NIC functionality, NICraft is applicable to commodity systems. By establishing a covert channel that bypasses on-system firewalls and detecting system activity like keystroke events, we highlight the feasibility and breadth of this threat—including local or offline scenarios. These findings underscore the need to reevaluate firmware-level security, especially as NICs becomes increasingly sophisticated. Future research should explore robust defenses, including methods to detect malicious firmware activity and stricter peripheral-OS isolation, to protect against this evolving class of side-channel attacks.

Acknowledgments

We thank Yepeng Pan, Till Schlüter, Manoj Gudi, and the anonymous reviewers for their valuable feedback and insightful suggestions.

References

1. Abel, A.: Cache replacement policy empirical study
2. Abel, A., Reineke, J.: nanoBench: A Low-Overhead Tool for Running Microbenchmarks on x86 Systems. In: ISPASS (2020)
3. AMD Inc.: AMD IOMMU Architectural Specification, Rev 2.00 (2011)
4. Blanco, A., Eissler, M.: One Firmware to Monitor 'Em All. In: Hack.Lu 2012. Core Security Technologies (2012)
5. Briongos, S., Malagón, P., Moya, J.M., Eisenbarth, T.: RELOAD+REFRESH: Abusing Cache Replacement Policies to Perform Stealthy Cache Attacks. In: USENIX Security (2020)
6. Chen, C.Y., Ghassami, A., Nagy, S., Yoon, M.K., Mohan, S., Kiyavash, N., Bobba, R.B., Pellizzoni, R.: Schedule-based side-channel attack in fixed-priority real-time systems (2015)
7. Chen, K.: Reversing and exploiting an Apple firmware update. Black Hat **69** (2009)
8. Choudhari, A., Guilley, S., Karray, K.: SpecDefender: Transient Execution Attack Defender using Performance Counters. In: ASHES (2022)
9. Duflot, L., Perez, Y., Morin, B.: What If You Can't Trust Your Network Card? In: RAID (2011)
10. Duflot, L., Perez, Y.A., Valadon, G., Levillain, O.: Can you still trust your network card. CanSecWest/core10 (2010)
11. Eran, H., Zeno, L., Tork, M., Malka, G., Silberstein, M.: NICA: An infrastructure for inline acceleration of network applications. In: USENIX ATC (2019)
12. Intel Corporation: Intel Resource Director Technology (Intel RDT) for 2nd Generation Intel Xeon Scalable Processors: Reference Manual
13. Intel Corporation: Intel Virtualization Technology for Directed I/O, Architecture Specification - Rev. 2.3 (2014)
14. Jiang, Q., Wang, C.: Sync+Sync: A Covert Channel Built on fsync with Storage. In: USENIX Security (2024)
15. Killourhy, K.S., Maxion, R.A.: Free vs. transcribed text for keystroke-dynamics evaluations. In: Proceedings of the LASER Workshop on Learning from Authoritative Security Experiment Results (2012)
16. Kocher, P., Horn, J., Fogh, A., , Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. In: (S&P'19) (2019)
17. Kosasih, W., Feng, Y., Chuengsatiansup, C., Yarom, Y., Zhu, Z.: SoK: Can We Really Detect Cache Side-Channel Attacks by Monitoring Performance Counters? In: ACM Asia CCS (2024)
18. Kurth, M., Gras, B., Andriesse, D., Giuffrida, C., Bos, H., Razavi, K.: NetCAT: Practical Cache Attacks from the Network. In: IEEE (SP) (2020)
19. Le, Y., Chang, H., Mukherjee, S., Wang, L., Akella, A., Swift, M.M., Lakshman, T.: UNO: Uniflying host and smart NIC offload for flexible packet processing. In: Symposium on Cloud Computing (2017)
20. Li, Y., McCune, J.M., Perrig, A.: VIPER: Verifying the Integrity of PERipherals' Firmware. In: CCS (2011)
21. Lipp, M., Gruss, D., Schwarz, M., Bidner, D., Maurice, C., Mangard, S.: Practical Keystroke Timing Attacks in Sandboxed JavaScript. In: ESORICS (2017)
22. Lipp, M., Schwarz, M., Raab, L., Lamster, L., Aga, M.T., Maurice, C., Gruss, D.: Nethammer: Inducing Rowhammer Faults through Network Requests. In: IEEE Euro S&P Workshops (2020)

23. Liu, F., Ge, Q., Yarom, Y., McKeen, F., Rozas, C.V., Heiser, G., Lee, R.B.: CATalyst: Defeating last-level cache side channel attacks in cloud computing. In: IEEE HPCA (2016)
24. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-Level Cache Side-Channel Attacks are Practical. In: IEEE Symposium on Security and Privacy (2015)
25. Liu, M., Peter, S., Krishnamurthy, A., Phothilimthana, P.M.: E3:Energy-Efficient microservices on SmartNIC-Accelerated servers. In: USENIX ATC (2019)
26. Markettos, A.T., Rothwell, C., Gutstein, B.F., Pearce, A., Neumann, P.G., Moore, S.W., Watson, R.N.M.: Thunderclap: Exploring Vulnerabilities in Operating System IOMMU Protection via DMA from Untrustworthy Peripherals. In: NDSS (2019)
27. Markuze, A., Vargaftik, S., Kupfer, G., Pismenny, B., Amit, N., Morrison, A., Tsafrir, D.: Characterizing, exploiting, and detecting DMA code injection vulnerabilities in the presence of an IOMMU. In: EuroSys (2021)
28. Maurice, C., Le Scouarnec, N., Neumann, C., Heen, O., Francillon, A.: Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters. In: RAID (2015)
29. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The Spy in the Sandbox: Practical Cache Attacks in JavaScript and their Implications. In: ACM CCS (2015)
30. Osvik, D.A., Shamir, A., Tromer, E.: Cache Attacks and Countermeasures: The Case of AES. In: Topics in Cryptology - CT-RSA (2006)
31. Schlüter, T., Choudhari, A., Hetterich, L., Trampert, L., Nemati, H., Ibrahim, A., Schwarz, M., Rossow, C., Tippenhauer, N.O.: Fetchbench: Systematic identification and characterization of proprietary prefetchers. In: ACM CCS (2023)
32. Schwarz, M., Schwarzl, M., Lipp, M., Masters, J., Gruss, D.: NetSpectre: Read Arbitrary Memory over Network. In: ESORICS (2019)
33. Stack Overflow: Enhanced rep movsb for memcpy (2017), `https://stackoverflow.com/questions/43343231/enhanced-rep-movsb-for-memcpy`
34. Stewin, P., Bystrov, I.: Understanding DMA Malware. In: DIMVA (2013)
35. Taram, M., Venkat, A., Tullsen, D.M.: Packet Chasing: Spying on Network Packets over a Cache Side-Channel. In: ACM/IEEE ISCA (2020)
36. Techarp: Cpu fast string bios option (2018), `https://www.techarp.com/bios-guide/cpu-fast-string/`
37. Triulzi, A.: Project Maux Mk.II: I Own the NIC, Now I Want a Shell. In: The 8th Annual PacSec Conference (2008)
38. Triulzi, A.: The Jedi Packet Takes Over the Deathstar: Taking NIC Backdoor to the Next Level. In: CanSecWest Conference (2010)
39. Vila, P., Ganty, P., Guarnieri, M., Köpf, B.: CacheQuery: learning replacement policies from hardware caches. In: ACM SIGPLAN (2020)
40. Vila, P., Köpf, B., Morales, J.F.: Theory and Practice of Finding Eviction Sets. In: IEEE (SP) (2019)
41. Wang, X., Shen, W., Bu, Y., Zhou, J., Zhou, Y.: DMAAUTH: A Lightweight Pointer Integrity-based Secure Architecture to Defeat DMA Attacks. In: USENIX Security (2024)
42. Zhang, T., Zhang, Y., Lee, R.B.: Cloudradar: A real-time side-channel attack detection system in clouds (2016)

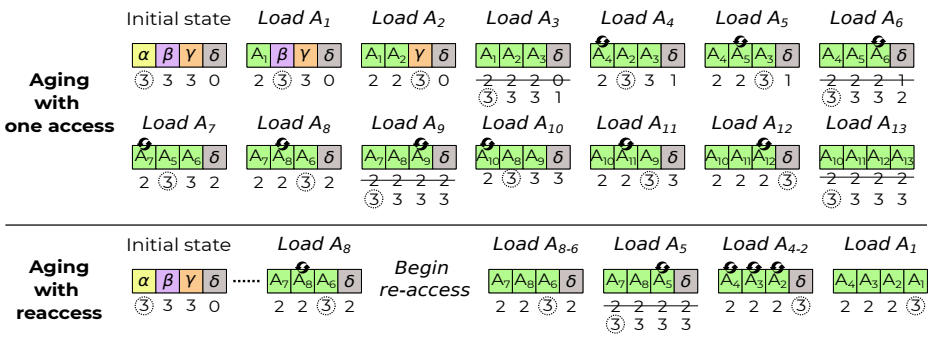


Fig. 11: *Aging* techniques: single access per packet (top) vs. re-accessing the same packet (bottom), where A_x denotes the *Aging* packet number x

A Fast-string optimization on x86 CPU

A technical challenge arises when aligning packets in the buffer. During the probe phase (via the *reassembly_ load* gadget), the *skb_ copy_ bits()* API copies data between fragment using *memcpy()*. When the source data is not cached, the CPU triggers a cache line miss, loading only the specific cache line needed. However, Intel's fast-strings optimization [36,33]causes the CPU to prefetch the next cache line, even if not needed. Consequently, a cache line miss also occurs when both the source and destination are cached, complicating PRIME+PROBE attacks.

Reverse-engineering on Intel Skylake CPUs reveals that this optimization is suppressed when $((dst < src) \wedge (src < 16))$, where src and dst are offsets within their respective cache lines during *memcpy()*. To meet this condition, we vary the IP header size using options fields, allowing header to share the cache line with the data. During priming (*checksum_ load* gadget), the CPU loads the header for checksum computation, ensuring the data is loaded into the target cache set.

B Aging with Reaccess

Figure 11 demonstrates the *Aging* process in a 4-way cache set. Initially, the cache holds lines with ages 3 (α), 3 (β), 3 (γ), and 0 (δ), with α as the next eviction candidate. In the single-access scenario (top), the attacker loads fragments sequentially: A_1, A_2, A_3, and so forth, until the cache is filled. α is replaced by A_1 (inserted at age 2), followed by $A_{2\text{-}6}$, progressively evicting older lines. Eventually, δ ages to 3 and becomes evictable. Loading A_{13} evicts δ, completing the *Aging* phase. Thirteen packets are needed—over three times the associativity—making eviction set construction non-trivial.

Re-accessing fragments significantly reduces this overhead. As shown in the bottom of Figure 11, the attacker loads eight fragments ($A_{1\text{-}8}$) via *checksum_ load*, then re-accesses them during *reassembly_ load* in reverse order. After re-accessing A_1, the set is fully aged. Thus, reaccess reduces the eviction set requirement to at most twice the associativity ($2N$), improving practicality.