MC^3: Memory Contention-based Covert Channel Communication on Shared DRAM System-on-Chips

Ismet Dagli
Department of Computer Science
Colorado School of Mines
Golden, CO, USA
ismetdagli@mines.edu

James Crea
Department of Computer Science
Colorado School of Mines
Golden, USA
jcrea@mines.edu

Soner Seckiner
Department of ECE
University of Rochester
Rochester, NY, USA
sseckine@ur.rochester.edu

Yuanchao Xu
Department of Computer Science
University of California Santa Cruz
Santa Cruz, CA, USA
yxu314@ucsc.edu

Selçuk Köse
Department of ECE
University of Rochester
Rochester, NY, USA
selcuk.kose@rochester.edu

Mehmet E. Belviranli
Department of Computer Science
Colorado School of Mines
Golden, CO, USA
belviranli@mines.edu

***Abstract*—Shared memory system-on-chips (SM-SoCs) are ubiquitously employed by a wide range of computing platforms, including edge/IoT devices, autonomous systems, and smartphones. In SM-SoCs, system-wide shared memory enables a convenient and cost-effective mechanism for making data accessible across dozens of processing units (PUs), such as CPU cores and domain-specific accelerators. Due to the diverse computational characteristics of the PUs they embed, SM-SoCs often do not employ a shared last-level cache (LLC). Although covert channel attacks have been widely studied in shared memory systems, high-throughput communication has previously been feasible only by relying on an LLC or by possessing privileged or physical access to the shared memory subsystem.**

In this study, we introduce a new memory-contention-based covert communication attack, MC^3, which specifically targets shared system memory in mobile SoCs. Unlike existing attacks, our approach achieves high-throughput communication without the need for an LLC or elevated access to the system. We explore the effectiveness of our methodology by demonstrating the trade-off between the channel transmission rate and the robustness of the communication. We evaluate MC^3 on NVIDIA Orin AGX, NX, and Nano platforms and achieve transmission rates up to 6.4 Kbps with less than 1% error rate.

I. Introduction

Mobile system-on-chips (SoCs) house multiple types of processing units (PUs), including general-purpose CPU cores and domain-specific accelerators (DSAs), such as GPUs and deep learning accelerators. With the proliferation of integrated DSAs, modern SoCs can provide cost-effective and energy-efficient execution, making them ideal candidates for in-the-field computing in many areas (mobile phones [36], smart home environments [16] and autonomous systems [25]).

An emerging architectural feature of modern SoCs (*e.g.*, NVIDIA's Orin [25], Apple's M3 [2], Qualcomm's Snapdragon [29]) is a shared main memory where the data is stored for access by all PUs. The use of shared physical memory (SM) in commodity SoCs is motivated by the goal of reducing the chip area and production costs. It can also provide additional performance benefits by minimizing data transfer overhead between the CPU and the DSAs [8], [7]. However, several studies [37], [15], [6] revealed that when running multiple workloads concurrently, PUs in SM-SoCs can experience significant slowdown caused by shared memory contention.

Over the years, security researchers have shown that having a shared hardware component with a predictable performance slowdown leaves a unique fingerprint. Using this fingerprint, various attacks have exploited cache [21], memory [39], storage [17], temperature [13], and power [33]. Covert channel communication attacks targeting vulnerabilities in the memory subsystem can be categorized into three: (1) *Cache-based, high-throughput attacks* which leverage the LLC between CPU cores [21], [38], [12], [20], [5], between multiple GPUs [10] and between a CPU and a GPU [9]. (2) *Low-throughput attacks targeting directly the DRAM* rely on memory performance attacks [31], memory deduplication [4], [35], bus snooping [39] and monitoring of DRAM power consumption [27]. (3) *Attacks requiring elevated privileges or hardware access* [28], [22], [31], [30]. None of these studies have shown how to build a fast, memory-contention-based covert channel without the need for privileged access (See Section III).

Constructing a fine-grained, low-noise, and high-throughput memory-contention-based covert channel attack on mobile SM-SoCs presents several challenges: (i) SM-SoCs generally lack a shared LLC to avoid complex design requirements. The trojan (*i.e.*, transmitter) needs to generate sufficient memory pressure that is observable by the spy (*i.e.*, receiver). CPU-based workloads in resource-limited SM-SoCs often fail to fully utilize the memory bandwidth even when all cores are used [37]. Therefore, accelerators with higher memory demands, such as GPUs, should be employed. Meanwhile, the generated memory pressure should be low enough to minimize the risk of being detected by system defenses. (ii) The total memory pressure exerted cumulatively by the spy and the trojan must be reliably high. Memory accesses satisfied by the caches can artificially

978-3-9826741-0-0/DATE25/© 2025 EDAA

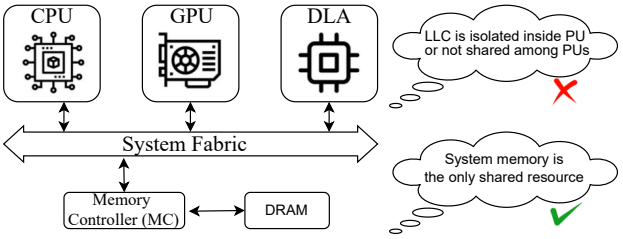

Fig. 1: Block diagram for NVIDIA's Xavier AGX SoC embedding a CPU, GPU, deep learning accelerator (DLA) and shared memory.

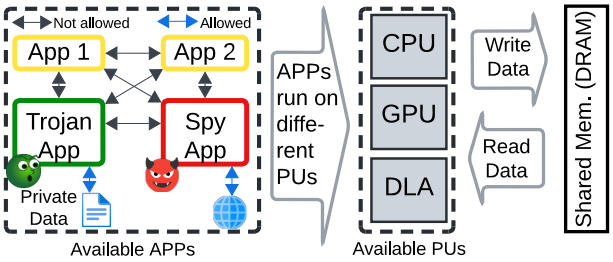

Fig. 2: Threat Model

increase perceived bandwidth, hence, they should be minimized to achieve a reliable access stream reaching the DRAM. This is also crucial to maximize the capacity of the communication channel. (iii) Without external synchronization mechanisms, reliable and high-throughput data transmission over SM becomes challenging, as the trojan and the spy may operate at different magnitudes of memory operations, particularly when located on different types of PUs (*e.g.*, CPU cores and GPU). (iv) Finally, these requirements should be achieved without elevated privileges and hardware access, and the attack should function under single-user and multi-application environments. Our proposed work addresses all four challenges listed above.

In this paper, we introduce a new memory contention-based covert communication attack, MC^3, targeting shared memory SoCs on mobile platforms. Our attack exploits the underlying vulnerability with software-only mechanisms, requiring neither direct hardware access nor super-user privileges. MC^3 is designed to achieve a balance between the communication accuracy and the transmission rate (*i.e.*, capacity) of the covert channel. To increase the transmission rate and improve the efficiency of our attack, we further propose a CPU+GPU version of the receiver and the transmitter. We demonstrate that MC^3 achieves transmission rates of up to 6.4 Kbps with an error rate below 1% in CPU-to-GPU communication. Our implementation is available at https://github.com/hypesys/MC3.

Our work makes the following contributions:

- We unveil a new attack vector that leverages the slowdown in memory accesses due to shared use of system memory. The attack vector is achieved through software-only measurements and does not require privileged access to the system.
- We present a novel covert channel attack that targets shared-memory SoCs without a last-level cache between its processing units. Our attack considers both CPU-GPU and CPU-CPU placements of the transmitter and the receiver.
- We evaluate MC^3 on NVIDIA Orin AGX, Nano, and NX SoCs and achieve a channel capacity of up to 6.4 Kbps with 95% accuracy. The accuracy reaches 99.99% when the capacity is capped at 1.3 Kpbs.

II. Background

A. Shared Memory SoCs (SM-SoC)

Modern SoCs, such as NVIDIA's Xavier and Orin architectures (as depicted in Fig. 1) integrate different types of accelerators, such as GPUs and DSAs, and each is optimized for specific computations. Unlike larger-scale systems where each accelerator has a dedicated primary memory, SM-SoCs share a common DRAM-based memory such as DDR4. Each PU has access to memory via a shared memory bus and a centralized memory controller (MC). Due to their inherent architectural heterogeneity, SM-SoCs often lack a shared LLC. Modern SoCs benefit from shared memory design because it reduces production costs and improves the data transfer overhead between CPU, GPU and other PUs.

B. Additional Related Work

Denial of service (DoS) attacks [24], [38] exhaust the memory subsystem and greatly increase memory access latency. Commonly implemented memory controller (MC) scheduling policies, such as fairness control [11] and adaptive scheduler [19], are typically designed to maximize system performance. Although MC schedulers that prioritize security [32], [34] mitigate memory performance attacks with limited overhead, they cannot fully eliminate the threat. Our work, however, focuses on covert-channel communication, which demands far higher precision than DoS attacks require.

Additional studies have shown how to create architectural covert channels in the cloud [39], [38], HPC servers [10], and desktops [23], [14]. Similar studies also used temperature [33], [13] and power [18], [33] as a covert communication channel. Our work focuses on the vulnerabilities stemming from shared memory use.

III. Threat Model

Figure 2 depicts our threat model which involves running two (or more) applications on an SM-SoC (as explained in Section II-A). The transmitter (*i.e.*, trojan) is an application that has access to sensitive or private user data. The receiver (*i.e.*, spy) is an application running on the same SM-SoC but does not have access to the same data. Applications running in the system (including the receiver and transmitter) are not allowed to communicate with each other. Both transmitter and receiver can run on the CPU or GPU (in no specific order) without elevated execution privileges —meaning they cannot access protected OS facilities or performance counters. The attack is designed to be executed remotely and attacker's presence or active engagement is not required. The attacker is assumed to have no physical access to the hardware components (*e.g.*, for measuring power consumption and electromagnetic emissions).

IV. Shared Memory Contention as Attack Vector

Our proposed methodology relies on the vulnerability that we discover in shared-memory SoCs and has the potential to be exploited on mobile and autonomous SoCs for a variety of attacks that do not require privileged access. A programmer can develop an adversarial transmitter application that leaves a distinct signature via purposeful shared memory accesses. This

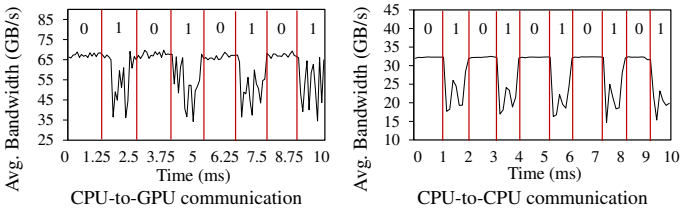

Fig. 3: Raw traces for CPU-to-GPU and CPU-to-CPU communication

signature can be used to leak crucial information that can be encoded in binary form.

Although this attack strategy seems similar to other types of covert channel attacks, there are unique challenges to efficiently and reliably designing shared memory contention channels:

- *Sufficiently observable contention:* While fully stressing memory resources to maximize contention is technically possible by the use of accelerators, the attack should generate sufficiently enough contention for the transmitter and receiver to communicate with each other. This will allow the attack to remain undetected by countermeasures deployed by the OS while maximizing the channel capacity of the attack. We deal with this challenge by creating contention that only targets shared memory resources and bypasses the private cache hierarchy of CPUs.
- *Reliable and efficient contention:* Achieving reliable contention generation necessitates a careful characterization of the channel's behavior. While reliability can be increased by repeatedly performing contention for a long time to transmit a bit, the practicality of the attack often requires minimizing the repetition. We overcome these challenges by thoroughly analyzing the contention behavior and using fine-grained time intervals for the receiver and the transmitter.
- *Synchronizing transmitter and receiver:* Unlike traditional cache attacks, where the effects of a cache hit-or-miss can be clearly observed in the order of nanoseconds, the slowdown caused by the memory contention becomes visible in the order of microseconds. This requires synchronized transmitter and receiver operation. Additionally, considering the differences in computational capabilities and clock rates of the CPUs and GPUs, the design of attack vectors on two diverse PUs requires the synchronization to be done without using any external resources. We overcome this challenge by developing a precise contention generator and sleep procedure for the transmitter and adapting them to the receiver accordingly.

Feasibility of shared memory contention-based covert channel: To demonstrate how the memory contention behavior affects the observed memory bandwidth of an application, we run the transmitter app on the CPU and the receiver on both the CPU and the GPU of an Orin NX SoC. Fig. 3 shows two raw traces of the varying average bandwidth (BW) perceived on the receiver side. Regardless of whether the receiver runs on the CPU or the GPU, the perceived BWs for the receiver have clear drops in the traces, which correspond to the '1's sent by the transmitter. This experiment demonstrates the feasibility of building covert channels with shared memory contention.

V. *MC*^{3}: Shared Memory Contention-based Covert Channel Communication

A. Overall Mechanism

Fig. 4 illustrates the communication protocol between the transmitter and receiver for transmitting bits (*i.e.*, 0 or 1) through shared memory contention. The transmitter conveys bits by performing buffer copy operations on memory while the receiver continuously performs another buffer copy operation to detect the transmitted bit.

The transmitter is responsible for sending the bit by modulating the level of memory contention. As shown in the upper part of Fig. 4, to transmit a bit '0', the transmitter sleeps for a predefined time interval of T_n. To send a bit '1', it performs continuous copy operations to access DRAM. The receiver continuously operates its own buffer copy function, then measures the duration of the buffer copy operation, and finally calculates the average BW over the duration, which will be used to decode the bit.

The lower part of Fig. 4 illustrates the case where the receiver can perform more copy operations with lower latency (*i.e.*, higher memory BW) while the transmitter is sleeping. Multiple copy operations per time interval T_n can also be utilized to have more reliable data transmission (see Sec. V-G). In contrast, when the transmitter induces contention by performing a copy operation, the receiver's throughput decreases (*i.e.*, lower memory BW) because memory latency increases due to shared memory contention.

While it is possible to run the receiver non-stop, we instead opt to start the receiver slightly earlier (*e.g.*, one second) than the transmitter at a predetermined epoch. This design decision eliminates the need for continuous operation of the receiver, thus minimizing the chances of our attack being detected under real-world conditions. The early start allows the receiver to collect BW information (*i.e.*, latency per copy operation) without any interference from the transmitter, which can be used as a baseline during the data analysis stage. Although other applications on the device may use shared memory — potentially introducing noise into the receiver's average BW measurements— the pattern of zeros and ones can be detected using heuristic history-based signal processing approaches.

Table I lists our test devices with varying computational capability and memory BW capacity. We use Jetpack 5.1 on all devices. It is worth noting that all devices have TrustZone Trusted Execution Environment (TEE) and OS-protection regions in the memory subsystem.

B. Attack Vector

Our attack leverages a memory-contention channel to stealthily transmit data between two processes.

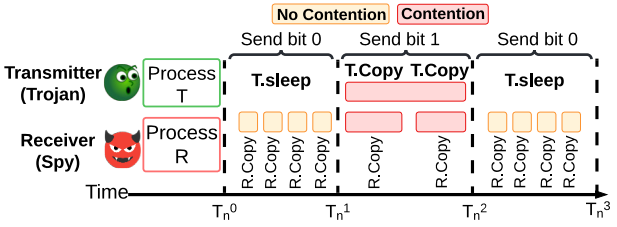

Fig. 4: Communication between the transmitter and receiver.

TABLE I: Targeted platforms

Device	Orin AGX	Orin Nano	Orin NX
CPU	12-core A78	6-core A78AE	8-core A78AE
GPU	2048 core Ampere	1024 core Ampere	1024 core Ampere
DRAM & BW	64 GB 256-bit 204.8 GB/s	8 GB 128-bit 68 GB/s	8 GB 128-bit 102 GB/s

Our transmitter algorithm, described in Alg. 1, is responsible for encoding the data and transmitting the encoded data bitstream over the memory-contention channel. It loops through the input data bitstream, either running the memory contention kernel (*i.e.*, data copy operation) for the specified duration T if the current bit is a 1, resulting in memory contention, or sleeping for the same duration T (if not set differently) if the current bit is a 0, resulting in near zero memory contention.

Our receiver algorithm, described in Alg. 2, is responsible for receiving the encoded data from the memory-contention channel and decoding it. It continuously runs the memory contention kernel over the duration T —the T value should be the same as the transmitter's. Then, it normalizes the BWs based on a global average (*i.e.*, subtract each BW sample from the overall average observed BW) [39]. Then, this normalized BW is thresholded with hysteresis, which we experimentally determined to be much more resistant to noise. Finally, the result of this thresholding is converted directly into the received bitstream, where the values above the threshold become 0 (which corresponds to a higher measured BW in the receiver, as a result of the transmitter *not* simultaneously generating memory contention) and the values below the additive inverse (*i.e.*, $-1\times$) of the threshold are 1 (which corresponds to a lower measured BW in the receiver, as a result of the transmitter simultaneously generating memory contention).

Algorithm 1 Transmitter

Input: data bitstream B and its length n, time interval T
for $i \leftarrow 0$ to $n-1$ **do**
  if $B[i]$ is 1 **then** CONTEND_FOR(T) ▷ Generate contention
  else SLEEP_FOR(T) ▷ Remain idle

Algorithm 2 Receiver

Input: hysteresis threshold γ, run length n, time interval T
$B \leftarrow []$ ▷ Output bitstream (starts empty)
$b \leftarrow 0$ ▷ Hysteresis state
$\bar{\beta} \leftarrow 0$ ▷ Average BW
for $i \leftarrow 0$ to $n-1$ **do**
  $\beta_{\text{raw}} \leftarrow$ CONTEND_FOR(T)
  $\bar{\beta} = \frac{\bar{\beta} \cdot i + \beta_{\text{raw}}}{i+1}$ ▷ Use simple global average
  $\beta_{\text{normalized}} = \beta_{\text{raw}} - \bar{\beta}$
  if $\beta_{\text{normalized}} > \gamma$ **then** APPEND(B, 0)
    $b \leftarrow 1$
  else if $\beta_{\text{normalized}} < (-1 \cdot \gamma)$ **then** APPEND(B, 1)
    $b \leftarrow 0$
  else APPEND(B, b)
return B

C. Cache-less Memory Access

Throughout the development of the memory-contention kernel, we observed that CPU caches are used to access the data and artificially inflate the memory BW measurements by using data-streaming kernels. Although sufficient contention generation using data streaming kernels is also possible, it makes our BW measurements unreliable and introduces substantial noise into the communication channel.

TABLE II: Precise 'contention' and 'sleep for' durations

Operation	Expected duration	Mean Error	Minimum Error	Maximum Error	Std. dev. Error
Sleep for	100 ms	46 ns	5 ns	314 ns	41 ns
Contention	100 ms	12 μs	5 ns	767 μs	65 μs

To alleviate this, our implementation employs memory instructions with non-temporal hints, specifically `ldnp`/`stnp` (Load/Store Pair Non-Temporal) Arm64 instructions. This hint signifies that the data being loaded or stored is unlikely to be reused soon, prompting the system to bypass the cache hierarchy [1]. By doing so, we ensure that our memory operations access DRAM directly, enhancing the reliability of our BW measurements and increasing the efficiency of the attack. It is worth noting that we also observed sufficient contention generation with data streaming using regular data streaming without non-temporal instructions.

D. Precise Contention Duration and Precise Sleep

In order to maximize performance, the sleep and data copy operations require high temporal precision (*i.e.*, sleep or run for the desired duration as accurately as possible). We implemented a precise sleep mechanism by utilizing an OS-provided function (*i.e.*, `std::this_thread::sleep_for`) until near the desired end time and finally spinning (*i.e.*, `while` loop with an empty body) until the desired end time is reached [3]. To implement `CONTEND_FOR(T)`, used in both algorithms, our data copy operation first runs the memory-contention kernel for a small, fixed amount of data (*i.e.*, taking nearly one second) to estimate the currently achievable memory contention BW (β_0). Using the total desired duration (T), it estimates the amount of data the kernel needs to run for (d_*, where $d_* \propto \beta_i \cdot T$). Using this estimate, it splits this total data estimate up (e.g.: $d_i = \frac{1}{100} \cdot d_*$), runs the memory-contention kernel (for d_i amount of data), collects β_i, updates d_* and d_i, and repeat. When it gets close to the desired end time, it further splits the estimate (e.g.: $d_i = \frac{1}{1000} \cdot d_*$) to reduce the amount of under/overshoot. In Table II, we report the results for 100 ms execution, demonstrating our average error rate of 4 and 7 orders of magnitude lower for sleep and contention generation, respectively.

E. Transmitter and Receiver Design

Our attack hinges on the transmitter generating sufficient memory pressure to create noticeable contention, which the receiver must detect. Essentially, the transmitter needs to generate enough memory access requests to contend with the receiver, and the receiver must be sensitive enough to observe the difference between the transmitter's sleep and data copy operations. To evaluate this, we conducted experiments on an Orin Nano using three CPU cores for both the transmitter and receiver. We send 1024 bits of information, evenly distributed with bits '0's and '1's, with the slowdown results illustrated in Fig. 5.a. We design varying buffer sizes of data copy for transmitter (from 0.6 MB to 128 MB) and receiver (from 1 MB to 100 MB). Overall, we observe an increasing pattern of average slowdown on the receiver side as we increase transmitter buffer sizes. For example, with a 1 MB buffer size for the receiver, the transmitter requires at least a 2 MB buffer

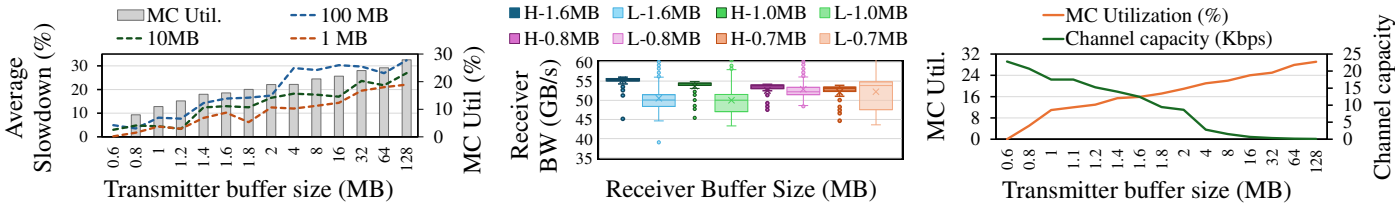

Fig. 5: (a) [Left] Average slowdown in the perceived BW depending on the transmitter buffer size. (b) [Middle] Average perceived high BW (H) and low BW (L) for bits '0' and '1', respectively, for varying receiver buffer sizes. (c) [Right] MC utilization per transmitter buffer size.

(*i.e.*, at least 20% MC utilization) to achieve a minimum of 10% slowdown.

Although we observe a slowdown while transmitting bit '0', we also need to clearly distinguish the BW differences between bit '1' and bit '0' on the receiver. To demonstrate this, we measure and analyze the average BW perceived on the receiver side while the transmitter is running, with a 4 MB buffer size sending '0' (higher perceived BW) and '1' (lower perceived BW) bits. Fig. 5.b depicts the distribution (including outliers) of the BW perceived by the receiver while sensing '0's (light colors) and '1's (dark colors). Although buffer sizes of 1.6 MB and 1.0 MB have clear differences in terms of perceived BW, lower buffer sizes for the receiver fail to distinguish the differences between bit '0' and '1' by looking at perceived BW. While outliers create noise if accuracy is calculated solely with average-based methods, history-based methods (which compare the current trace with the previous) clearly identify the changes. Although the transmitter with 0.8 MB buffer has approximately 10% MC utilization, contention may not be enough to distinguish bit '0's and '1's.

F. Trade-off between Copy Duration and Contention Amount

The channel capacity intuitively depends on the size of the transmitter's buffer that is being copied. Assuming that, to transmit bit '1', transmitter copy operation with a fixed buffer size will be performed once during time interval T_n, there exists an inverse relationship between the transmitter buffer size and the channel capacity, as demonstrated in Eq. 1. As we increase the transmitter buffer size, thus the time $Time_{Tra}$ to copy the buffer, and account for the average slowdown $Slowdown_{Rec}$ perceived by the receiver, the channel capacity decreases.

$$Channel\ Capacity = 1\ /\ (Time_{Tra} * Slowdown_{Rec}) \quad (1)$$

Ideally, we aim to use the smallest possible transmitter buffer size to maximize the channel capacity of our covert channel. On the other hand, there is a lower limit to how much we can decrease the buffer size to generate observable contention. To demonstrate the trade-off between channel capacity and buffer size, we vary the transmitter buffer size and report the results in Fig. 5.c. We observe that increasing the transmitter buffer size leads to a near-linear decrease in channel capacity. For example, with transmitter buffer sizes of 0.5 MB and 0.6 MB, we achieved channel capacities of up to 25 Kbps, yet the receiver was unable to detect the transmitter's activity since the transmitter's MC utilization was nearly zero.

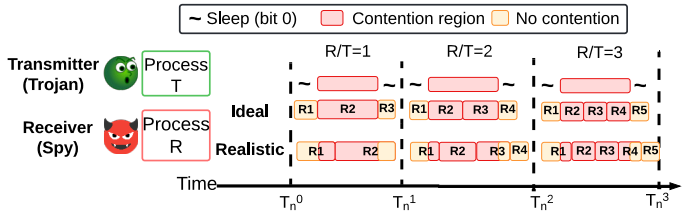

Fig. 6: The overlap between [T]ransmitter's and [R]eceiver's copy operations for various R/T copy epoch ratios. 'Ideal' represents the expected durations and 'actual' represents the observed. R1-R5 indicates the epoch number of a copy operation performed by the receiver.

G. Reducing Noise

Synchronizing transmitter and receiver is essential for accurately sensing BW differences. Since direct communication between the transmitter and receiver is not allowed (which would otherwise defeat the point of a covert channel), we must statically decide the time intervals. However, as depicted in Fig. 5.b, due to contention generation being inherently noisy, the accumulation of outlier-induced errors can lead to desynchronization. If we aim to operate in the worst-case scenario, then many contended regions may complete earlier than anticipated.

We illustrate and compare the expected (*i.e.*, ideal) and observed (*i.e.*, actual) copy operation durations in Fig. 6. *R/T* denotes the ratio of copy epochs (or iterations) that the (R)eceiver performs for every bit '1' sent by the (T)ransmitter. When *R/T = 1*, the receiver perceives the contention from the transmitter with a delay, causing a drop in observed BW and increasing in noise Conversely, when the transmitter sends bit '1' across multiple epochs (*i.e.*, *R/T > 1*), the receiver will capture at least one or more fully contended regions. This substantially improves transmission accuracy, while reducing the communication capacity of the covert channel.

VI. Improving Channel Capacity with GPUs

Mobile and autonomous SoCs often embed GPUs which are designed to enable massive parallelism. This results in better utilization of the memory subsystem compared to the CPUs.

A. Receiver on the GPU

The receiver must generate sufficiently high BW to accurately distinguish between a '0' bit and a '1' bit. To achieve better accuracy, we run the receiver on the GPU and the transmitter on the CPU. We implement the memory copy operation with

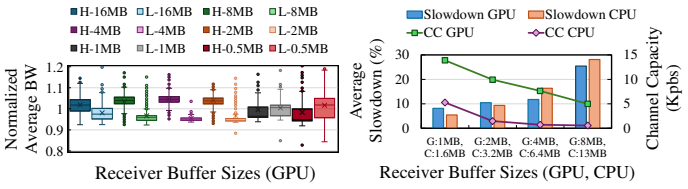

Fig. 7: (a) [Left] The distribution of the perceived BW distribution for varying buffer sizes. H and L indicates '0' and '1' transmissions, respectively. (b) [Right] Slowdown in the perceived BW and channel capacity when receiver is on (C)PU and (G)PU.

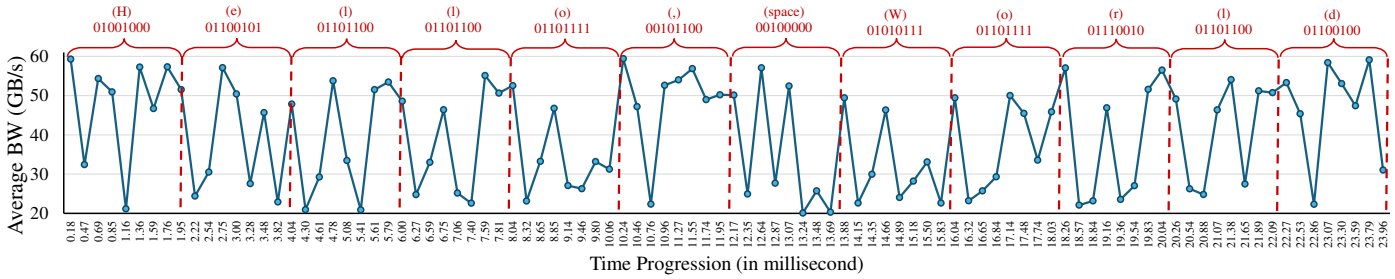

Fig. 8: Average BW observed by the receiver while receiving a "Hello, World" message.

`cudaMemcpy` using CUDA [26]. Fig. 7.a shows the average receiver slowdown for the '0' (*i.e.*, high) and '1' (*i.e.*, low) bits using a buffer size of 1 MB on an Orin AGX. For receiver buffer sizes from 2MB to 8MB, we observe a clear distinction between '0's and '1's. However, buffer sizes of 0.5MB and 1MB fail to sense the contention, whereas buffer sizes of 16MB and beyond may result in a misalignment of the copy epochs as explained in Sec. V-G. We also experimented with configuring the transmitter on the GPU and the receiver on the CPU. However, unlike the case where the receiver is on the GPU, placing the transmitter on the GPU did not improve the distinction of '1' and '0' bits. These results were omitted because of space limitations.

B. Channel Capacities for Receivers on the GPU and CPU

To further understand the relationship between channel capacity and the slowdown observed when the receiver is placed on CPU and GPU, we perform an experiment on Orin AGX where we gradually increase receiver buffer sizes and report the results in Fig. 7.b. We observe that we can achieve channel capacities of up to 14 Kbps and 5 Kbps on GPU and CPU, respectively, with 9% and 7% slowdowns observed in the measured BW. As we increase buffer sizes, we typically observe less channel capacity but more sensible contention on both CPU- and GPU-based receivers. It is worth noting that, when the receiver runs on GPU, we can map the transmitter to 11 cores out of the 12 available CPU cores. This mapping significantly increases the contention generation capacity of the transmitter. Overall, the GPU-based receiver achieves approximately 3× – 5× higher channel capacities on Orin AGX (and 2× – 4× on Orin Nano) compared to the CPU-based receiver.

C. Hello World Transmission

To assess our design's performance with longer messages, we transmit a 100 Kb text message on Orin Nano and observe how the perceived BW changes over time. The results are depicted in Fig. 8. The y-axis shows the rolling average of perceived BW during each time interval, whereas the x-axis represents the time progression in milliseconds. The initial portion of the message is transmitted and received with 100% accuracy while the entire message is delivered with 99.02% accuracy at a channel capacity in excess of 4 Kbps.

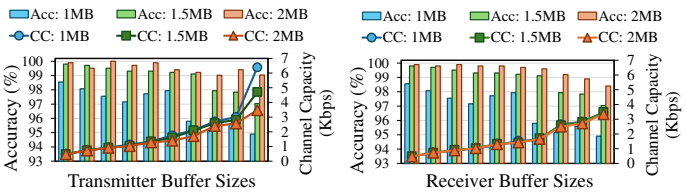

Fig. 9: The trade-off between accuracy and channel capacity for varying (a) transmitter [left] and (b) receiver buffer sizes [right].

D. Channel Capacity vs. Transmission Accuracy

As the final overarching experiment, we vary the transmitter and buffer sizes and observe the resulting trade-off between channel capacity and transmission accuracy when the receiver is run on the GPU and the transmitter on the CPU of Orin AGX. In Fig. 9.a, we varied the transmitter buffer sizes while keeping the receiver buffer size fixed at 1 MB. We increase the channel capacity by varying the number of copy epochs/iterations of the buffer copy operation per time interval from 1 to 10, and adjusting the receiver accordingly. In general, our results demonstrate that MC^3 achieves either up to 6.4 Kbps channel capacity or up to 99.99% transmission accuracy. As the transmitter buffer size and the number of copy iterations per interval increased, we observe higher accuracy but reduced channel capacity. Some notable data points are:

- The 2MB transmitter buffer size achieves 99. 1% accuracy at 3.5 Kpbs channel capacity and a near-perfect accuracy of 99.99% at 1.3 Kpbs.
- The 1MB transmitter buffer size maximizes channel capacity up to 6.4 Kpbs while achieving a decent 94.9% accuracy.

In Fig. 9.b, we increase receiver's buffer size and buffer copy operation iterations per interval, but keep the transmitter buffer size constant at 1 MB. Overall, increasing the receiver buffer size (with a constant transmitter copy iteration) improved accuracy with minimal impact on channel the capacity. It is worth noting that increasing the receiver size degrades the accuracy since *R/T* ratio becomes unbalanced once the buffer size is 5 MB and beyond. Similar to the observations in Fig. 9.a, increasing the channel capacity by decreasing the transmitter copy operations per interval leads to a decrease in accuracy.

VII. Conclusion

In conclusion, we demonstrate a novel and efficient covert channel attack that exploits shared-memory contention in SM-SoCs. The proposed attack does not require privileged access to the system and achieves a channel bandwidth of up to 6.4 Kbps with accuracy rates that reach 99.99%. We unveil an important vulnerability that could be used to leak private data in modern mobile and autonomous systems.

Acknowledgements

This work is supported in part by the National Science Foundation (NSF) under Grant No. CNS-2350228. Any opinions, findings, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Arm® Architecture Reference Manual for A-profile architecture, 2024.

[2] Apple. Apple unveils the new macbook pro featuring the m3 family of chips, 2023. (accessed on 09/22/2024).

[3] Blat Blatnik. The perfect Sleep() function. https://blog.bearcats.nl/perfect-sleep-function/.

[4] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE symposium on security and privacy (SP)*, pages 987–1004. IEEE, 2016.

[5] Yun Chen, Arash Pashrashid, Yongzheng Wu, and Trevor E Carlson. Prime+ reset: Introducing a novel cross-world covert-channel through comprehensive security analysis on arm trustzone. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[6] Ismet Dagli and Mehmet E Belviranli. Shared memory-contention-aware concurrent dnn execution for diversely heterogeneous system-on-chips. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP'24)*, pages 243–256, 2024.

[7] Ismet Dagli, Alexander Cieslewicz, Jedidiah McClurg, and Mehmet E Belviranli. Axonn: Energy-aware execution of neural network inference on multi-accelerator heterogeneous socs. In *DAC*, pages 1069–1074, 2022.

[8] Mohammad Dashti and Alexandra Fedorova. Analyzing memory management methods on integrated cpu-gpu systems. In *Proceedings of the 2017 ACM SIGPLAN International Symposium on Memory Management (ISMM)*, pages 59–69, 2017.

[9] Sankha Baran Dutta, Hoda Naghibijouybari, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. Leaky buddies: Cross-component covert channels on integrated cpu-gpu systems. In *ISCA*, pages 972–984. IEEE, 2021.

[10] Sankha Baran Dutta, Hoda Naghibijouybari, Arjun Gupta, Nael Abu-Ghazaleh, Andres Marquez, and Kevin Barker. Spy in the gpu-box: Covert and side channel attacks on multi-gpu systems. In *ISCA*, pages 1–13, 2023.

[11] Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. *ACM Sigplan Notices*, 45(3):335–346, 2010.

[12] Aditya S Gangwar, Prathamesh N Tanksale, Shirshendu Das, and Sudeepta Mishra. Flush+ early reload: Covert channels attack on shared llc using mshr merging. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[13] Jeferson González-Gómez, Kevin Cordero-Zuñiga, Lars Bauer, and Jörg Henkel. The first concept and real-world deployment of a gpu-based thermal covert channel: Attack and countermeasures. In *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2023.

[14] Marc Green, Leandro Rodrigues-Lima, Andreas Zankl, Gorka Irazoqui, Johann Heyszl, and Thomas Eisenbarth. {AutoLock}: Why cache attacks on {ARM} are harder than you think. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1075–1091, 2017.

[15] Mark Hill and Vijay Janapa Reddi. Gables: A roofline model for mobile socs. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 317–330. IEEE, 2019.

[16] Ahmad Jalal, Majid Ali Khan Quaid, and Kibum Kim. A wrist worn acceleration based human motion analysis and classification for ambient smart home system. *Journal of Electrical Engineering & Technology*, 14:1733–1739, 2019.

[17] Qisheng Jiang and Chundong Wang. Sync+ sync: A covert channel built on fsync with storage. *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[18] S Karen Khatamifard, Longfei Wang, Amitabh Das, Selçuk Köse, and Ulya R Karpuzcu. Powert channels: A novel class of covert communicationexploiting power management vulnerabilities. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 291–303. IEEE, 2019.

[19] Yoongu Kim, Dongsu Han, Onur Mutlu, and Mor Harchol-Balter. Atlas: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1–12. IEEE, 2010.

[20] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. {ARMageddon}: Cache attacks on mobile devices. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 549–564, 2016.

[21] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy (S&P)*, pages 605–622. IEEE, 2015.

[22] Yang Luo, Wu Luo, Xiaoning Sun, Qingni Shen, Anbang Ruan, and Zhonghai Wu. Whispers between the containers: High-capacity covert channel attacks in docker. In *2016 IEEE trustcom/bigdatase/ispa*, pages 630–637. IEEE, 2016.

[23] Nikolay Matyunin, Nikolaos A Anagnostopoulos, Spyros Boukoros, Markus Heinrich, André Schaller, Maksim Kolinichenko, and Stefan Katzenbeisser. Tracking private browsing sessions using cpu-based covert channels. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks (WiSeC)*, pages 63–74, 2018.

[24] Thomas Moscibroda Onur Mutlu. Memory performance attacks: Denial of memory service in multi-core systems. In *USENIX security*, 2007.

[25] NVIDIA. Next-level ai performance for next-gen robotics | nvidia jetson orin agx. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/, 2023. (accessed on 09/22/2024).

[26] NVIDIA. Cuda samples. https://github.com/NVIDIA/cuda-samples/tree/master/Samples/1_Utilities/bandwidthTest, 2024. (accessed on 09/22/2024).

[27] Thales Bandiera Paiva, Javier Navaridas, and Routo Terada. Robust covert channels based on dram power consumption. In *Information Security: 22nd International Conference, ISC 2019, New York City, NY, USA, September 16–18, 2019, Proceedings 22*, pages 319–338. Springer, 2019.

[28] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. {DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks. In *25th USENIX security symposium (USENIX security 16)*, pages 565–581, 2016.

[29] Qualcomm. Snapdragon mobile platforms. https://www.qualcomm.com/snapdragon/products, 2024.

[30] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. Zombieload: Cross-privilege-boundary data sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (ASIA-CCS)*, pages 753–768, 2019.

[31] Michael Schwarz, Clémentine Maurice, Daniel Gruss, and Stefan Mangard. Fantastic timers and where to find them: High-resolution microarchitectural attacks in javascript. In *Financial Cryptography and Data Security: 21st International Conference, FC 2017, Sliema, Malta, April 3-7, 2017, Revised Selected Papers 21*, pages 247–267. Springer, 2017.

[32] Ali Shafiee, Akhila Gundu, Manjunath Shevgoor, Rajeev Balasubramonian, and Mohit Tiwari. Avoiding information leakage in the memory controller with fixed service policies. In *Proceedings of the 48th International Symposium on Microarchitecture (MICRO)*, pages 89–101, 2015.

[33] Hritvik Taneja, Jason Kim, Jie Jeff Xu, Stephan Van Schaik, Daniel Genkin, and Yuval Yarom. Hot pixels: Frequency, power, and temperature attacks on {GPUs} and arm {SoCs}. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 6275–6292, 2023.

[34] Yao Wang, Andrew Ferraiuolo, and G Edward Suh. Timing channel protection for a shared memory controller. In *HPCA*, pages 225–236. IEEE, 2014.

[35] Jidong Xiao, Zhang Xu, Hai Huang, and Haining Wang. Security implications of memory deduplication in a virtualized environment. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

[36] Yunyang Xiong, Hanxiao Liu, Suyog Gupta, Berkin Akin, Gabriel Bender, Yongzhe Wang, Pieter-Jan Kindermans, Mingxing Tan, Vikas Singh, and Bo Chen. Mobiledets: Searching for object detection architectures for mobile accelerators. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, pages 3825–3834, 2021.

[37] Yuanchao Xu, Mehmet Esat Belviranli, Xipeng Shen, and Jeffrey Vetter. Pccs: Processor-centric contention-aware slowdown model for heterogeneous system-on-chips. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1282–1295, 2021.

[38] Tianwei Zhang, Yinqian Zhang, and Ruby B Lee. Dos attacks on your memory in cloud. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security (CCS)*, pages 253–265, 2017.

[39] Wu Zhenyu, Xu Zhang, and H Wang. Whispers in the hyper-space: high-speed covert channel attacks in the cloud. In *USENIX Security symposium*, pages 159–173, 2012.