Prime+Reset: Introducing A Novel Cross-World Covert-Channel Through Comprehensive Security Analysis on ARM TrustZone

Yun Chen*,[1], Arash Pashrashid*,[1,2], Yongzheng Wu[2] and Trevor E. Carlson[1]
[1]National University of Singapore
[2]Huawei Research Center
{yun.chen,pashrashid.arash}@u.nus.edu, wu.yongzheng@huawi.com, tcarlson@comp.nus.edu.sg

***Abstract*—ARM TrustZone, a robust security mechanism, aims to protect against a wide range of threats by partitioning the system-on-chip hardware and software into two distinct worlds, namely the normal world and the secure world. However, the secure world still remains susceptible to malicious attacks, including side-channel and covert-channel vulnerabilities.**

Previous efforts to leak data from TrustZone focused on cache-based and performance monitoring unit (PMU)-based channels; in this paper, we, however, propose a security analysis benchmark suite by traversing the hardware components involved in the microarchitecture to study their security impact on the secure world. Our investigation unveils an undisclosed leakage source stemming from the L2 prefetcher. We design a new cross-core and cross-world covert-channel attack based on our reverse engineering of the L2 prefetcher, named Prime+Reset. Compared to most cross-world covert-channel attacks, Prime+Reset is a cache- and PMU-agnostic attack that effectively bypasses many existing defenses. The throughput of Prime+Reset can achieve 776 Kib/s, which demonstrates a significant improvement, 70×, over the state-of-the-art, while maintaining a similar error rate (< 2%). One can find the code at https://github.com/yunchen-juuuump/prime-reset.

I. Introduction

With the relentless proliferation of connected devices and the ever-expanding digital landscape, security has emerged as a major concern in the realm of embedded systems and mobile computing platforms. ARM TrustZone [21], an innovative hardware security technology, plays a key role in enhancing the security of modern processors against a multitude of threats. TrustZone partitions the system into a secure world and a normal world, allowing for the execution of trusted code in isolation from regular applications. While TrustZone has contributed to system security, it is not resilient to certain malicious attacks, most notably side-channel attacks [15], [25] and covert-channel [7] attacks.

Side-channel attacks adeptly exploit observable characteristics, such as timing information, to illicitly extract cryptographic secrets or sensitive data. On the other hand, covert channel attacks exploit unintended information leakage through legitimate communication channels, enabling secret data transmission between the normal world and the secure world.

*These two authors contributed equally.

The need for a comprehensive security analysis of side channel and covert channel attacks on ARM TrustZone becomes apparent when we consider the rapid spread of ARM-based devices across a diverse range of applications, from mobile phones and Internet of Things (IoT) devices to critical infrastructure systems [4].

In this work, we explore the vulnerabilities that threaten the security of ARM TrustZone by conducting an in-depth analysis of side-channels and covert-channels. These comprehensive investigations shed light on the fine-tuned aspects of TrustZone's microarchitectural components and their susceptibility to exploitation. While prior research has predominantly centered on cache-based and PMU-based side channels and proposed mitigation strategies [8], [13], [23], our work extends to a broader array of hardware components that may be shared between the secure and normal worlds. We explore the ARMv8 processors' microarchitecture, and our investigation reveals a previously undisclosed vulnerability that leaks information across the world isolation boundary via the L2 prefetcher.

A key aspect of this new vulnerability is that it bypasses all mitigations proposed for cache-based and PMU-based channels. In other words, most existing mitigation strategies *cannot* offer the comprehensive security guarantee that ARM TrustZone demands in light of our discoveries. The main contributions of this work are:

- A security analysis benchmark suite that allows users to automatically mark potential leaks on their ARMv8 processors. This suite provides a comprehensive side-channel and covert-channel vulnerability analysis for ARMv8 TrustZone (see Section III).
- A reverse-engineering suite for analyzing the newly detected potential leakage source, the L2 prefetcher on the ARMv8 processor family. The suite characterizes the L2 prefetcher's critical features for building a new covert-channel attack (see Section IV).
- A novel cross-core and cross-world covert-channel, named Prime+Reset, which neither relies on the cache nor the PMU. Compared with the state-of-the-art cross-world covert-channel, Prime+Reset provides a significant throughput improvement of 70× while keeping a low error rate (see Section V).

II. Background and Related Works

A. Out-of-Order Processor

An out-of-order (OoO) processor leverages instruction-level parallelism (ILP) through the execution of instructions out of program order. The processor can schedule independent instructions for execution as soon as their dependencies are resolved, regardless of the original program sequence or memory fetch order. We explain the key components of OoO processors:

Frontend. The processor's frontend fetches and decodes instructions from memory, predicts branches using the Branch Prediction Unit (BPU), and dispatches them to an instruction scheduler.

Reorder Buffer. To ensure correctness and maintain transparency for the user, these processors employ a Reorder Buffer (ROB), which functions as a First-In-First-Out (FIFO) queue. The ROB tracks the status of each instruction within the pipeline, from dispatch to instruction commit.

Backend. To schedule multiple independent instructions for execution out-of-program-order, the backend of an out-of-order processor implements an instruction scheduler in the Issue Unit that controls the instruction issue. Instructions are typically stored in different slots based on their instruction type (e.g., memory, integer, floating-point), and the scheduler monitors every slot and issues instructions for execution as soon as their dependencies are resolved and an appropriate execution unit is available. An instruction's dependencies are resolved when its input operands become available.

Translation Lookaside Buffer (TLB). The TLB is a hardware cache that accelerates the translation of virtual memory addresses to physical memory addresses, making memory access faster and more efficient.

Prefetcher. A hardware component that anticipates future memory access needs by fetching data from main memory into the cache before it's actually requested, thus reducing memory latency.

Cache System. The cache stores frequently accessed data and instructions for rapid access, reducing memory latency. In scenarios where an address is not present in the L1 cache, the processor proceeds to look up the data in the shared next-level cache (e.g., L2 cache). If the requested data is still not found, and this is the last cache level, a memory request is then sent to the DRAM.

B. ARM TrustZone Architecture

ARM TrustZone [21] is a hardware-based security framework embedded in ARM processors, dividing the system into two isolated domains: the secure world and the normal world. This framework establishes a Trusted Execution Environment (TEE) in the secure world, protecting critical operations like cryptographic key storage and secure authentication.

TrustZone employs hardware-driven isolation mechanisms, including secure memory partitions and privilege levels, to maintain a strict separation between the secure and normal worlds. The strict partition allows both trusted and untrusted applications to coexist while protecting sensitive data and system functions.

TABLE I
Comparison with State-of-the-art Covert-Channel on ARMv8 TrustZone. **AO**, **SM**, and **PMU** represent agnostic, shared memory, and Performance Monitoring Unit respectively.

Vulnerability	Hardware Leakage	Cross Core	PMU AO.	Cache AO.†	SM AO.
μarch-Count [18]	μarch events	✗	✗	○	✓
Prime+Count [7]	Cache events	✓	✗	●	✓
Prime+Probe [17]	L2 Cache	✓	✓	●	✓
Directory Prime+Probe [16]	Cache Directory	✓	✓	◐	✓
Flush+Evict [15]	L2 Cache + PMU	✓	✗	●	✗
Prime+Reset (this work)	L2 Prefetcher	✓	✓	○	✓

†: ○ no cache needed; ◐ cache coherence required; ● cache required

C. Side-Channel and Covert-Channel Attacks on TEEs

Side-channel and covert-channel attacks represent critical security concerns in Trusted Execution Environments (TEEs). Side-channel attacks exploit information leaked through system characteristics like execution time differences, potentially revealing sensitive data [5], [15], [17]. Covert-channel attacks use unintended communication pathways within TEEs to stealthily transmit data, compromising isolation [7].

Table I summarizes recent side-channel and covert-channel attacks on TrustZone. Prime+Count builds covert channels in both single- and cross-core scenarios within the TrustZone architecture by tracking the number of occupied cache sets or lines [7]. Other recent works [15]–[17] monitored L2 cache and cache directory behavior to design their attacks on ARM processors. Li et al. [18] introduce a covert channel that is built and trained by reading PMU data from user space. They model the PMU footprint created through secure world execution while executing normal world workloads. Each of these attacks relies on a cache system or PMU. Furthermore, legitimate channels across worlds, including parameters passed by registers and direct shared memory read/write operations, cannot leak information now as they are protected by access control [13].

Defenses have been proposed against side-channel and covert-channel attacks on TrustZone [8], [13]. However, they mainly focus on cache-based or PMU-based channels and do not offer further analysis for other structures that can be shared between the secure and normal worlds. To the best of our knowledge, a comprehensive analysis of TrustZone's side-channel and covert-channel vulnerabilities has not been conducted previously and none of the proposed mitigations considered the TrustZone prefetcher a potential source of data leakage.

III. Security Analysis

In this section, we introduce our security benchmark suite, designed to automatically identify potential hardware side-channel vulnerabilities. Specifically, drawing insights from existing side-channel and covert-channel attacks, we recognize two essential criteria for a successful side-channel/covert-

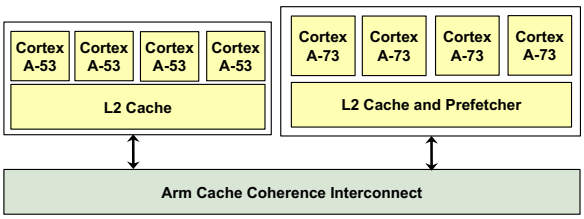

Fig. 1. Microarchitecture of Kirin 960 System-on-Chip.

channel attack. First, such an attack requires and uses shared hardware resources across different execution domains to create resource contention and detect variations (e.g., cache, BPU, Issue Unit, and TLB). Second, any microarchitectural events capable of altering the normal execution flow can introduce new leaks (e.g., out-of-order execution, and prefetcher).

Hence, our suite initiates its evaluation by detecting whether any hardware component serves as a shared resource or temporally changes the execution flow. Upon identification, we flag it as a potential source of leakage and subsequently conduct an in-depth analysis, including reverse-engineering, to understand its characteristics.

A. Test platform

In this study, we utilize a commercial ARM hardware platform, the HiKey960 featuring the Kirin 960 SoC [2], as a demonstration of our security benchmark suite's effectiveness. Notably, our security analysis suite is microarchitecture-agnostic (i.e., we do not rely on detailed information about the hardware) and is compatible with any ARMv8-based processor. The microarchitecture of the HiKey960, as depicted in Figure 1, contains four Cortex A53 cores (small core cluster) and four Cortex A73 cores (big core cluster). The L2 cache is shared within each cluster, and cross-cluster communication and synchronization are facilitated through ARM's cache coherence interconnection (CCI). We run a Linux 4.19 kernel as a normal world OS, and OP-TEE 3.19 [22], an open-sourced ARM TrustZone design, as the trusted OS.

B. Branch Prediction Unit

Our first microbenchmark, focused on the Branch Prediction Unit (BPU) [9], [14], seeks to determine if it's shared between the secure and normal worlds. To achieve architecture-agnostic results, we duplicate an application containing a branch from the secure world into the normal world[1], ensuring consistent Program Counter (PC) values[2]. In the normal world, we train the branch with a consistent *taken* direction. Following training, we switch to the secure world to execute the branch with a *not-taken* direction. If the BPU is shared between different worlds, the secure world's application would experience a branch misprediction, requiring more time to resolve the branch than when executing the *taken* direction. However, we observed that both directions exhibit similar resolution times. This observation strongly suggests that the BPU trained in the normal world is not utilized by the secure world, confirming that the BPU is not shared between the secure world and the normal world[3].

C. Backend

While out-of-order execution offers substantial performance benefits, it also introduces security concerns [10], [11]. In our second benchmark, we investigate whether the out-of-order execution engine compromises isolation between the secure and normal worlds. We examine potential TLB and Issue Unit contention. We find that the latency of accessing a prior cached page in the normal world is consistent with a latency close to that of the off-chip memory access (i.e., TLB miss) after the world switch, indicating that the TLB is consistently flushed after a world switch. Furthermore, we observed that the execution time of the normal world application, such as loop counting, remains consistent regardless of the type of workload executed in the secure world. These observations suggest that the Issue Unit is not shared between the two worlds. These results demonstrate that the entire backend pipeline is flushed upon world switching rendering the backend immune to these types of side-channel/covert-channel attacks.

However, Meltdown-type attacks [3], [20] exploit delays between exceptions (e.g., accessing kernel space from user space) to conduct microarchitectural data sampling attacks without context switching. To assess the secure world's vulnerability to Meltdown-type attacks, our benchmark conducts a Meltdown attack [20]. We discovered that the security monitor always checks process privilege before accessing secure world memory, conclusively demonstrating that the secure world can defend against Meltdown-type attacks.

D. L1 Cache and Prefetcher

Modern ARM processors commonly suffer from cache attacks [15], [17], [19]. To assess whether cache sharing occurs between the normal and secure worlds, our benchmark suite attempts to evict a target cache line from the normal world within the secure world's context.

In the first step, we load various cache lines into different L1 cache sets[4]. Following this, we switch to the secure world and introduce a delay before switching back to the normal world to inspect the cache. Our experimental findings indicate that all cache lines within the L1 data cache are invalidated after the world switch, even when the secure world application remains idle. We also performed similar tests on the L1 prefetcher, yielding identical results.

E. L2 Prefetcher

The above results collectively affirm the complete isolation of the same physical core between the secure and normal worlds when the PMU is not used. Comparatively, when running the secure and normal worlds in parallel on two separate cores, there is an increased likelihood of information leakage from the

[1]Various methods, like `fork` or `clone`, can be used for process duplication.

[2]Note that we do not employ kernel address space layout randomization (KASLR) or address space layout randomization (ASLR).

[3]The reason could be either the use of branch prediction isolation or flushing, which we will study in detail in our future work.

[4]Because L1 data cache comes in either 2-way or 4-way set-associative configurations in ARMv8 processors, we generate data blocks for both settings.

TABLE II
LEAKAGE DETECTION ON ALL POTENTIAL LEAKAGE SOURCES. NOTE THAT WE HAVE EMBEDDED PREVIOUS MICROBENCHMARKS IN OUR END-TO-END BENCHMARK SUITE TO ENABLE USERS TO AUTOMATICALLY DETECT ALL POTENTIAL LEAKAGES IN THEIR HARDWARE PLATFORM.

Microbenchmark	Location	Hardware	Leakage
[7] [18]*	In-Core	PMU	Prime+Count μarch-Count
[17]	Un-Core	L2 Cache	Prime+Probe
[16]	Un-Core	Coherence Directory	Dir. Prime+Probe
[15]	Un-Core	PMU	Num. of Eviction events on shared memory
This Work	In-Core	BPU	✗
This Work	In-Core	Backend	✗
This Work	In-Core	L1 Prefetcher	✗
This Work	In-Core	L1 Cache	✗
This Work	Un-Core	L2 Prefetcher	New leakage

secure world. Recent works [15]–[17], [19] have demonstrated that certain un-core hardware components, such as the L2 cache and ARM-CCI, are shared among different cores and worlds.

To further explore potential new sources of leakage within un-core hardware, our benchmark suite analyzes the Extended Control Register (ECTLR), which governs both in-core and un-core systems. Through our analysis and ARM Cortex-A doucmentation [1], we have uncovered the presence of an L2 stride prefetcher within the ARMv8 Cortex-A out-of-order processor family, enabled by default. Given that the L2 prefetcher resides within the un-core memory system and has not been previously well-documented in prior work, we identify it as a novel potential source of leakage and conduct an in-depth analysis through reverse-engineering.

Table II provides an overview of all the benchmarks within our security analysis benchmark suite.

IV. REVERSE-ENGINEERING L2 PREFETCHER

In this section, we explore into the L2 stride prefetcher's[5] characteristics, revealing index, update, and trigger mechanisms for the first time. Based on this newly revealed information, we then explore cross-core prefetching effects and determine the prefetcher's entry count to build a cross-core and cross-world covert-channel attack.

A. Triggering

A stride prefetcher begins fetching memory addresses when it detects a repeated pattern of memory access with a specific stride. To determine the exact trigger threshold, we designed a microbenchmark (see Listing 1) where different `step` values generate varying lengths of L2 cache misses in strided memory streams for prefetcher training. After training, we time the expected prefetched cache line. Figure 2 shows that the prefetcher starts prefetching after detecting the stride pattern three times.

B. Stride Update Policy

To better understand the prefetcher's stride update mechanism, we conducted tests by accessing `mem[offset]` using the trained memory function after training the prefetcher, followed

[5]Though coupled with the L1 prefetcher, we can easily remove L1 prefetcher noise as data prefetched by L2 prefetcher has higher latency.

```
1  char* probe = mmap(NULL, PAGESIZE, ..);
2  int step = atoi(argv[1]), stride = 5;
3  for (int i = 0 ; i < step; i++)
4          memory_access_1(probe + i * stride);
5  Time(mem[step * stride]);
```

Listing 1: Microbenchmark for detecting the triggering threshold in the L2 prefetcher.

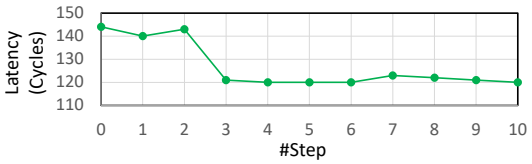

Fig. 2. Results of if the prefetcher is triggerable.

by timing the access to `mem[offset + stride]` (see Listing 2). Our findings reveal that `mem[offset + stride]` is prefetched only if `offset - last_address = stride`. This observation reveals that the prefetcher consistently updates the stride prior to initiating prefetching.

```
1  train_prefetcher(); //Function in the Listing 1
2  memory_access_1(probe + offset);
3  Time(mem[probe + offset + (step + 1) * stride]);
```

Listing 2: Microbenchmark for determining the stride update policy in the L2 prefetcher.

C. Index

Previous studies [5], [6] have explored how prefetchers can be indexed, either by the program counter (PC) or virtual page address. To test the PC's role in indexing, we modified line 2 in Listing 2 with a new memory instruction that matched the last N bits of the trained PC. However, even with $N = 0$, the prefetcher was still triggered, indicating that the PC does not determine the prefetcher's indexing. Additionally, we observed different PCs updating the stride on the same page, confirming that the stride prefetcher relies on page address indexing.

D. Entries

To determine the total number of prefetcher entries, we trained the prefetcher on N different pages and assessed whether the first page remained triggerable. As illustrated in Figure 3, it becomes evident that the first entry is evicted as soon as an additional 10 primed entries are introduced. Considering this observation, it is concluded that the prefetcher has 10 entries.

E. Interplay with other cores

The previous benchmarks focused on characterizing the L2 prefetcher's structure within a single core. To assess if the L2 prefetcher can be considered a real leakage source, we investigated its interaction with different cores. In our study, we set up two processes, named `A` and `B`, running on separate cores. The execution sequence of these processes is depicted in Figure 4. Initially, `A`, operating in the normal world, primes

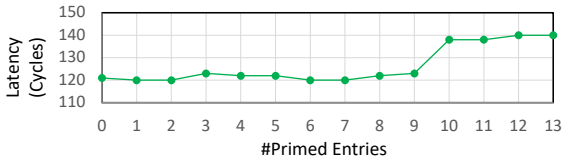

Fig. 3. Results of the number of entries of L2 prefetcher.

TABLE III
TEST RESULT OF WHETHER PROCESS B CAN INTERFERE WITH PROCESS A VIA L2 PREFETCHER.

Process B		Process A	
Location	**Behavior**	**Behavior**	**Result**†
Same cluster	Primes 10 entries	Check prefetcher	✓
Same cluster	Gen. SIGSEGV	Check prefetcher	✗
Same cluster	Gen. other signals	Check prefetcher	✓
Different clusters	Gen. SIGSEGV	Check prefetcher	✓
Same cluster	Gen. SIGSEGV	Time cached data	●

†: ✓: Triggerable prefetcher, ✗ : Non-triggerable prefetcher, ●: Cached data

a prefetcher entry and subsequently introduces a delay. Concurrently, B functions in the secure world, priming 10 entries in the L2 prefetcher. A ultimately evaluates whether the trained entry is evicted by B by checking the prefetcher status[6]. We, however, observed no eviction, even when we increased the primed entries to 20 in B and ran it in the normal world. This shows that the L2 prefetcher is statically partitioned among different cores, with each core having 10 entries.

Although the L2 prefetcher is not directly shared between cores, we identified a critical feature that can lead to leakage: the L2 prefetcher status resets when any core in the same cluster generates a `SIGSEGV` signal, typically from actions like accessing unallocated memory or manual triggering with `raise()` function call. We ruled out other signals as triggers. Importantly, the resulting `SIGSEGV` doesn't clear cached data; it only affects the prefetcher status.

Table III summarizes our findings, revealing that one core's entries in the L2 prefetcher status can be influenced by another core in the same cluster only when the latter generates a `SIGSEGV`. We hypothesize that this reset is a hardware feature of ARMv8, unrelated to OS-launched privileged instructions. Given that cached data remains unaffected, and as these processes don't share memory, we hypothesize that this feature could relate to memory consistency for maintaining the correctness of memory data but not cache coherence.

V. A CROSS-CORE AND CROSS-WORLD CACHE-AGNOSTIC COVERT-CHANNEL VIA L2 PREFETCHER

Based on the reverse-engineering findings of the L2 prefetcher, this section introduces the development of a cross-core and cross-world covert-channel attack, termed Prime+Reset. Prime+Reset serves as a means for transmitting data between the secure world and the normal world. Notably, Prime+Reset operates independently of the cache system or PMU, signifying it could bypass various existing mitigation techniques [13], [17], [23], [24].

[6]We utilized lines 2 and 3 in Listing 2 and configured the `offset` to `last_address + stride` for prefetcher status checking.

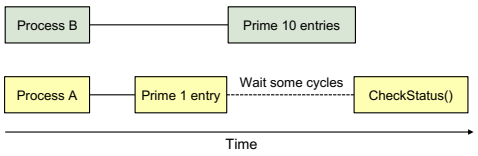

Fig. 4. Execution flow of Process A and B. Note that A and B are running on different cores.

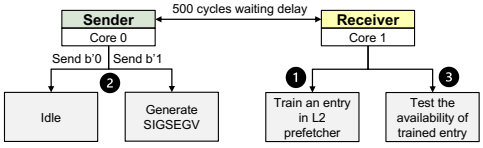

Fig. 5. Attack flow of Prime+Reset

A. Threat Model

Our attack assumes a scenario where a sender operates in either the secure or normal world, intending to transmit data to a receiver in a different world in the same cluster but on a different core[7]. They do not share memory or have direct communication. To synchronize their activities, refer to prior covert-channel attacks [7], [12], they use timing-fixed delays, and predefined protocols for error correction. We implement a high-resolution timer with a new kernel driver, but it is worth noting that prior research [19] has explored user-space methods to achieve this without needing OS-level privilege, thereby eliminating the Prime+Reset's necessity for OS-level control.

B. Prime+Reset Attack Flow

The attack process is shown in Figure 5. The receiver primes an L2 prefetcher entry and waits for a set number of cycles. When the sender wants to transmit '1', it generates a `SIGSEGV` signal; otherwise, it stays idle. We've added a signal handler to prevent program termination, allowing the process to continue after the signal for a better throughput. Furthermore, as the generation and propagation of the `SIGSEGV` signal require some cycles, a short delay on the receiver's end might lead to the prefetcher status being checked before the signal arrives, potentially increasing the error rate. Conversely, an excessively long delay would negatively impact throughput. To balance accuracy and throughput, we found that a 500-cycle delay is optimal. After this delay, the receiver checks the prefetcher status to determine if the sender sent '1' or '0'.

C. Attack Scenarios and Results

In cross-world covert-channel communication, data is often transmitted from the secure world to the normal world [7], [18] because sensitive data primarily resides in the secure world. We also explore scenarios where communication flows from the normal world to the secure world, such as when a secure world receiver awaits a command from a normal world sender.

[7]In this paper, we focus on the more significant cross-core scenario; however, Prime+Reset is also applicable to the same-core scenario.

TABLE IV
COMPARISON WITH STATE-OF-THE-ART CROSS-WORLD COVERT-CHANNEL ATTACKS. **S** AND **N** REPRESENT THE SECURE WORLD AND THE NORMAL WORLD REPRESETIVELY. † THIS WORK.

	Scenario	Error Rate	Max. Throughput
Prime+Count [7]	S to N	lower 1%	∼1 Kib/s
μarch-Count [18]	S to N	lower 1%	12 Kib/s
Prime+Reset†	S to N	1.8%	776 Kib/s
Prime+Reset†	N to S	lower 1%	776 Kib/s
Prime+Reset†	N to N	lower 1%	776 Kib/s

Additionally, we investigate a channel within the normal world since Prime+Reset does not rely on OS capabilities. Figure 6, Figure 7, and Figure 8 illustrate the outcomes of these scenarios using the transmission message 0x89abcdef. We exclude the secure world to the secure world scenario as secure world applications already have the highest privilege.

Table IV provides an overview of Prime+Reset's throughput and error rates, compared with previous cross-world covert-channels. Notably, Prime+Reset maintains consistent throughput across scenarios due to its uniform synchronization mechanism. It achieves a 70× improvement in throughput, over the state-of-the-art [18], while maintaining similar accuracy.

The main reason for this significant improvement in throughput is the robustness of the L2 prefetcher used in Prime+Reset. Unlike previous works, which depended on collecting various events from PMU and, potentially, the noise seen in the L2 cache of the multi-core system, Prime+Reset doesn't require complex synchronization methods. Instead, it uses a more noise-resistant L2 prefetcher, resulting in better capability.

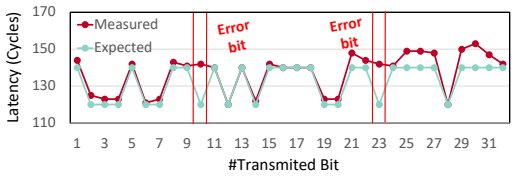

Fig. 6. Transmit data from the secure world to the normal world.

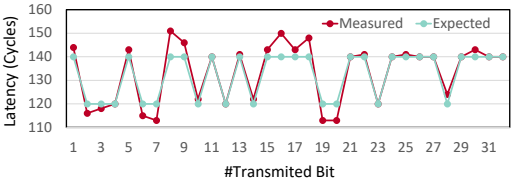

Fig. 7. Transmit data from the normal world to the secure world.

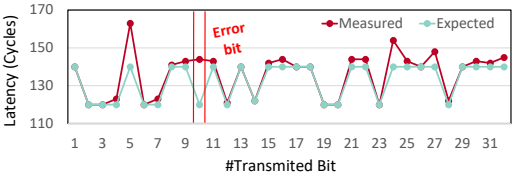

Fig. 8. Transmit data from the normal world to the normal world.

VI. CONCLUSION

In this work, we develop an automated benchmark suite to identify potential hardware leakage that may cause a cross-world side-channel/covert-channel on ARMv8 processors. Our analysis revealed that the unstudied L2 prefetcher may be a new leakage source. We then propose a reverse-engineering suite to uncover the L2 prefetcher's characteristics, revealing the `SIGSEGV` signal's impact on the prefetcher status. Leveraging this, we design a cache- and PMU-agnostic cross-core and cross-world covert-channel attack called Prime+Reset that achieves a 70× throughput improvement over state-of-the-art methods while maintaining accuracy.

To mitigate Prime+Reset, one option is to disable the L2 prefetcher, which, although it may impact performance, will effectively eliminate the attack.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their detailed feedback which allowed us to improve this work. This work was supported by a grant from the National Research Foundation (NRF) of Singapore (NRF2018NCR-NCR002).

REFERENCES

[1] "Arm Cortex-A73 MPCore processor technical reference manual r1p0," https://developer.arm.com/documentation/100048/0100/level-1-memory-system/memory-prefetching/data-prefetching-and-monitoring.
[2] "Hikey960 board," https://www.96boards.org/product/hikey960.
[3] C. Canella *et al.*, "Fallout: Leaking data on meltdown-resistant CPUs," in *CCS*, 2019.
[4] D. Cerdeira *et al.*, "Sok: Understanding the prevailing security vulnerabilities in TrustZone-assisted tee systems," in *SP*, 2020.
[5] Y. Chen *et al.*, "AfterImage: Leaking control flow data and tracking load operations via the hardware prefetcher," in *ASPLOS*, 2023.
[6] Y. Chen *et al.*, "New cross-core cache-agnostic and prefetcher-based side-channels and covert-channels," *arXiv preprint arXiv:2306.11195*, 2023.
[7] H. Cho *et al.*, "Prime+Count: Novel cross-world covert channels on Arm TrustZone," in *ACSAC*, 2018.
[8] G. Dessouky *et al.*, "HybCache: Hybrid side-channel-resilient caches for trusted execution environments," in *USENIX Security*, 2020.
[9] D. Evtyushkin *et al.*, "BranchScope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, 2018.
[10] S. Gast *et al.*, "Squip: Exploiting the scheduler queue contention side channel," in *SP*, 2023.
[11] B. Gras *et al.*, "Translation leak-aside buffer: Defeating cache side-channel protections with TLB attacks," in *USENIX Security*, 2018.
[12] Y. Guo *et al.*, "Leaky Way: A conflict-based cache covert channel bypassing set associativity," in *MICRO*, 2022.
[13] J. S. Jang *et al.*, "Secret: Secure channel between rich execution environment and trusted execution environment." in *NDSS*, 2015.
[14] P. Kocher *et al.*, "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, 2020.
[15] Z. Kou *et al.*, "Load-Step: A precise TrustZone execution control framework for exploring new side-channel attacks like flush+evict," in *DAC*, 2021.
[16] Z. Kou *et al.*, "Attack directories on Arm big.LITTLE processors," in *ICCAD*, 2022.
[17] Z. Kou *et al.*, "Cache side-channel attacks and defenses of the sliding window algorithm in TEEs," in *DATE*, 2023.
[18] X. Li *et al.*, "Cross-world covert channel on Arm TrustZone through PMU," *Sensors*, 2022.
[19] M. Lipp *et al.*, "ARMageddon: Cache attacks on mobile devices," in *USENIX Security*, 2016.
[20] L. Moritz *et al.*, "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, 2020.
[21] B. Ngabonziza *et al.*, "TrustZone explained: Architectural features and use cases," in *CIC*, 2016.
[22] "OP-TEE," https://www.trustedfirmware.org/projects/op-tee/.
[23] A. Pashrashid *et al.*, "HidFix: Efficient mitigation of cache-based spectre attacks through hidden rollbacks," in *ICCAD*, 2023.
[24] N. Zhang *et al.*, "CaSE: Cache-assisted secure execution on Arm processors," in *S&P*, 2016.
[25] N. Zhang *et al.*, "Trusense: Information leakage from TrustZone," in *INFOCOM*, 2018.