Cost and Effectiveness of TrustZone Defense and Side-Channel Attack on ARM Platform

Naiwei Liu[1]*, Meng Yu[2]†, Wanyu Zang[2], and Ravi Sandhu[1]
[1]Institute for Cyber Security, University of Texas at San Antonio, San Antonio TX 78249, USA
[2]Roosevelt University, Chicago IL 60605, USA

Received: November 2, 2020; Accepted: December 11, 2020; Published: December 31, 2020

Abstract

Security concerns on ARM platform have been developing in recent years, with some security design and implementations being introduced on ARM platform. As ARM structure is developing into ARMv8 version, some security research and design had been applied into recent chips. For example, TrustZone applies to security concerns of users with ARM Cortex-A and Cortex-M series chips, providing secure and private enclaves. However, the security design on ARM is severely challenged by different type of attackers. Side-channel attack is one of the major threats to ARM platform with TrustZone. In this paper, we have discussions on the performance and overhead of TrustZone and cache-related instructions, and some stats of side-channel attack. Our experimental and theoretical evaluations can help in design of defense framework based on ARM TrustZone, and provide evidence of how efficient FLUSH operations can work in defense against cache threats.

Keyworkd: ARM TrustZone, System Security, Side-Channel Attack

1 Introduction

In recent years, ARM platform and ARM security have been a new research focus. Security frameworks designed and implemented are usually utilize the new feature on ARM since ARMv7: TrustZone. TrustZone is a secure enclave provided by ARM on both Cortex-A and Cortex-M series. These defense frameworks target to memory protection, process protection and even cache protection. For example, some of the malicious users can utilize the entry/exit of the TrustZone on ARM Cortex-A, launching a cache-based attack, and compromising the message channel between victim threads and the system. As a result, some research papers target to this problem using access control of entry/exit operations, and some papers use isolated cache protection design. The research papers and their implementations can cut down the bandwidth of cache-based attack, with various level of overhead on the whole system.

On the attacker side, research efforts have been on the Internet of Things (IoT) and IoT devices. Papers introducing threats to systems and ARM chips have been published in a rising trend. Cache in these devices becomes the research focus on both single device environment and cloud with multiple devices, or even IoT network connecting smart devices. The attacks can be very effective on extracting the users' private and secured data, without the permissions and access to the protected enclaves. Side-channel attack among them is a research focus. Malicious hackers can collect performance data, power consumption data or even some 'trash' data to try retrieving useful information. Attackers derive users'

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 11(4):1-15, Dec. 2020
DOI:*10.22667/JOWUA.2020.12.31.001*

*Co-corresponding author: Institute for Cyber Security, University of Texas at San Antonio, San Antonio TX 78249, USA, Tel: +1(210)343-0856, Email: liunaiwei32@gmail.com

†Co-corresponding author: Roosevelt University, Chicago IL 60605, USA, Tel: +1(804)399-2252, Email: myu04@roosevelt.edu

information like cryptographic keys, protected or private data by launching attack on the cache, and analyze the information from what they get. Some attackers just try to collect the difference in access time with different memory blocks, and predict what is accessed frequently by the users. The difference in access time can be collected if the attacker and the victims are sharing data in the cache.

Security design and defense framework on ARM can be very different from traditional x86 structure. On ARM, it has multiple privilege levels and some instructions are privileged. On ARM Cortex M3/M4 chips, there is no cache in these chips, and main memory is fully mapped with a special place being set as 'cachable' part. The design is basically for the utilization situation of ARM Cortex-M devices as they are always powered by limited sources, even batteries. On the other hand, ARM devices are always asked to work in some mobile situations with long battery durations. In this case, the performance limitation is something that we have to consider.

Facing cache threats like side-channel attack, cache FLUSH operations are useful because they can affect attackers' time measurement phase. However, these operations can affect the system performance as well. To design based on FLUSH and other cache operations, we must consider the balance in performance overhead, security statistics and other measurements. We have some experiments on the balance of that, and we have adaptive defense designs in other papers. This paper is extended of our WISA 2020 conference paper [14], with side-channel implementations and more experiments added into the paper discussions. We added side-channel implementations and discussions, making the paper not only talk about cost and performance on the defense side, but at attacker side as well. We also have some review on the recent research papers in Related Work chapter.

In this paper, we investigate the defense effectiveness to cache based side-channel attacks on the ARM architecture. We design several tests based on TrustZone on both ARM cortex-A and cortex-M series chips and get the performance data. These can help in design and implementation of defense, while keeping the performance and effectiveness balanced. Overall, we have following contributions in this paper:

- We measure the overhead of TrustZone related instructions;
- We implemented side-channel attack instances on ARM platform and show how practical it can be in real life;
- We measure FLUSH operation overhead and analyze clock cycles they take on different platforms;
- We provide the best/worst case of defense performance based on our experimental results and analysis.

The structure of this paper is as follows: in Related Work section, we introduce previous research and recent research on this topic, analyzing their strong contribution and weaknesses; in Overview section, we introduce our environments of development, structure of design and security assumptions; in Implementation section, we provide some details about our design and experiment settings; in Evaluation section, we provide experimental results and discussion; and in Conclusion section we have our conclusions on the research topic.

2 Related Work

2.1 Cache-Based Attack

In a cloud computing system or a computer with multiple processes and threads, the Last Level Cache (LLC) is shared among multiple processor cores, making it vulnerable to LLC based side-channel attacks. Unlike L1 cache, LLC is much slower than L1 cache, leading to more difficult set up for side

channels. There are different ways to launch side-channel attacks, e.g., FLUSH+RELOAD [21], [7], PRIME+PROBE [7] [9] [13], and bus-locking [20].

For example, the FLUSH+RELOAD involves three steps. The attacker first flushes one or more of the desired cache contents using processor-specific instructions (e.g. clflush on x86 processors). Second, the attacker waits for sufficient time for the victim to use (or not to use) the flushed cache area. Finally, the attacker reloads previously flushed cache lines, measuring the reload time for each one of them to infer if it was touched by the victim. FLUSH+RELOAD strategy has been proven very effectively in many side channel attacks on x86 architecture. For example, Gulmezoglu et al. [7] recovered the AES key of OpenSSL within 15 seconds. Yarom and Falkner [21] recover a RSA encryption key across VMware VMs using FLUSH+RELOAD attack, and Irazoqui et al. [10] recovered AES keys using similar attack and exploiting the vulnerabilities in cache. For PRIME+PROBE attack, Work [13] recover AES keys in a cross-VM Xen 4.1 using PRIME+PROBE attack. Liu et al. [12] presented a PRIME+PROBE type side-channel attack model against the LLC, which is tested to be practical and threatens the system.

2.2 Defense Summary

Bernstein [2] suggested to add L1-table-lookup instruction to load an entire table in L1 cache, and also load a selected table entry in a constant number of CPU cycles. Page [15] investigated a partitioned cache architecture. Wang and Lee [19] [18] [17] proposed new security-aware cache designs to thwart the LLC side channel attack with low overhead. In [18], the Partition-Locked cache (PLcache) was able to lock a sensitive cache partition into cache, and Random Permutation cache (RPcache) randomized the mapping from memory locations to cache sets. In [12], a novel random fill cache architecture that replaces demand fetch with random cache fill within a configurable neighborhood window was proposed. While the hardware solutions provide strong isolations between the victim and the attacker, they require special hardware features that are not immediately available form commodity processors. Some researchers proposed to modify applications to better protect secrets from side-channel attacks. Brickell et al. [3] proposed three individual mitigation strategies: compact S-box table, frequently randomized tables, and pre-loading of relevant cache-lines. It compressed and randomized tables for AES. However, it requires manually rewriting the AES implementation and is specific to AES. Cleemput et al. [4] applied the mitigating code transformations to eliminate or minimize key-dependent execution time variations. Crane et al. [5] proposed a software diversity technique to transform each program unique. The approach offers probabilistic protection against both online and off-line side-channel attacks. In their work, using function or basic-block level dynamic control-flow diversity along with static cache noise results in a performance slowdown of 1.76x-2.02x compared to the baseline AES encryption when using 10%-50% cache noise insertion. Dynamic cache noise at 10%-50% has significantly impact on performance (2.39-2.87x slowdown). However, above software solutions are typically application specific or incur substantial performance overhead.

In summary, recent papers on the defense are into two major categories: hardware-based solutions and software-based solutions. We summarize the design highlights, features and weakness in following table:

2.3 Recent Research on ARM Defenses

On the year 2017, Sandro Pinto and some other researchers proposed LTZVisor [16], which is based on TrustZone to protect and assist ARM virtualization. They implement and test on ARM platform and have an overhead of around 22% at the highest user switching frequency. Guan el al. proposed TrustShadow [6], using TrustZone to protect user's applications, with little or no change on the application itself. The overhead here is around 10% at worst case and 2% on average. However, the framework

Table 1: Related Work on Defense Design against Cache Threats

Paper	Feature	Hardware/Software	Weakness
Bernstein et al., 2005	L1-table-lookup instruction	Hardware	Only on x86 platform
Page et al., 2005	Partitioned cache architecture		Performance concerns; complexity
Liu, Lee et al., 2014	Random cache-fill architecture		Not adaptive to multiple devices
Brickell et al. 2006	Individual mitigation strategies	Software	Only on x86 platform
Cleemput et al., 2014	Mitigating code transformations		Application-specific
Crane et al., 2015	Probabilistic protection		Incurring overhead and not so effective

is not tested on ARMv8-M, which has different structure and instruction sets from ARM Cortex-A series. Similar as LTZVisor, Hua et al. designed and implemented vTZ [8], a virtualization based defense framework on ARM. The overhead of vTZ is on average case around 5%.

According to their work, the most popular solution on ARM is virtualization, using TrustZone to protect the application, data and user's private keys. This can only be implemented on ARM Cortex-A series, which has different level of cache, multiple privilege levels and powerful CPU. On ARMv8-M series, however, similar implementation is not applicable. On ARMv8-M series chips, there is no cache on the structure, and normally the protection cannot be complicated due to the limited resource on the devices. Compared with their work, we have a more directly protection, with acceptable overhead and good performance.

2.4 Recent Research on ARM TrustZone

In recent years, some papers have discussions and new research findings on ARM platform, especially focusing on TrustZone protection. Zhang et al. [22] proposed an Android protection framework using TrustZone on ARM, protecting VoIP phone calls. It enclaves privacy data so the phone calls cannot be intercepted easily by malicious eavesdropping. Amacher et al. [1] have evaluate the performance of ARM TrustZone using TEEs and different benchmarks, but the security concern is out of that paper's scope. Keystone defense framework proposed by Dayeol Lee and others [11] is a good example of defense framework based on TrustZone. It enclaves protected operations and disables sharing in TLBs and memory blocks so there's no side-channel attack based on the vulnerability here. However, the timing side-channel attack is out of that paper's scope. In our discussion, there are still risks of side-channels when exiting from TrustZone, so we need also investigate the vulnerability at the gate of security enclave.

We also have a paper based on the effectiveness of TrustZone: On the Cost-Effectiveness of TrustZone Defense on ARM Platform, which is published on the 21st World Conference on Information Security Applications (WISA '20) [14]. In this paper, we added with more experiments and discussions added and several evaluations. We discuss the cost and performance on the defense side with TrustZone assist, and also implement side-channel experiments to show how it works on the attacker side, making the discussion more completed and systematic.

3 System Architecture and Environment

3.1 Background

As multi-core processors become pervasive and the number of on-die cores increases, a key design issue facing processor architects is the security layers and policies for the on-die LLC. With LLC techniques, a CPU might only need to get around 5% data from main memory, which can improve the efficiency of CPU largely. In our implementations, we are using Intel i7-4790 processor, with 8Mb SmartCache. On ARMv8 Cortex-A platform, we are using Juno r1 Development Platform which has one A57 and one

A53 processors on the board. A57 has a 2M LLC on the processor. On Cortex-M platform, we are using ARM Cortex-M4 series chips, the development platform has 3 pipeline stages and no built-in cache.

With the increasing complexity of computing systems, as well as multiple level of memory access, some registers are designed to store some specific hardware events. These registers are usually called hardware performance counters. We have many tools getting information from those performance counters, thus getting the performance information.

In our implementation, we use perf to collect the execution information of the programs. However, we cannot use perf for collecting timing information of memory access, since it cannot be accurate enough. On this paper we use inline assemblies and consult some related registers to measure time associated information with our side-channels.

3.2 Design on ARM Cortex-A

According to our evaluation on current on-the-market systems and applications, we find out that more and more Trusted Execution Environment (TEE) technologies are being used on the implementations of secure system. Besides, most of the implementations are utilizing ARM TrustZone to protect the memory access and critical data. As we are interested in the performance overhead of defending using FLUSH operations on exiting TrustZone, the experiments should start from the measurements of using TrustZone, like the time cost and performance overhead.

Our experiments on ARM Cortex-A are in three different steps. For the first step, we test the cost of entering and exiting from TrustZone. After we get the exact data (clock cycles) related to TrustZone, the next step is to measure how much it takes up for the TEEs to call TrustZone related instructions or operations. On the third step, we try to clean the cache every time the system exiting from TrustZone, and see the performance overhead by these FLUSH operations added to the system. As the cache gets FLUSHed every time after the using of TrustZone, the risk of being side-channel attacked can be theoretically cut down to non-exist.

3.3 Overview on ARM Cortex-M

Unlike ARM Cortex-A series chips, M-series chips have different structure, and with other limitations. Most IoT devices are based on Cortex-A platform, but still a rising trend that more products are using Cortex-M platform. As a result, it is still valuable to investigate the defense against malicious attackers with TrustZone. In this paper, we have similar tests on ARMv8-M platform, measuring the performance of TrustZone, as well as FLUSH operation overhead. Our experiments on Cortex-M are using ARM Versatile V2M-MPS2 Motherboard with ARM Cortex-M4 cores. It offers 8Mb of single cycle SRAM, and 16Mb of PSRAM. It supports the application of different ARM Cortex-M classes, from Cortex-M0, to M3, M4, and M7. Besides these support, the development board supports simulation of ARMv8-M.

As mentioned above, on Cortex-M4 series chips, there is no built-in cache. However, the memory structure on M4 is different from other structures like x86 and Cortex-A. On that platform, memory blocks are allocated in fixed order, taking their assigned responsibilities. It is quite different from dynamic allocation, and is to the consideration of power consumption and performance overhead. Among these memory blocks, some are acting as 'cache-in-memory', so we can still see them working like cache and operate some instructions to read the working status of it.

The experiments are in two different steps. First, we measure the time cost entering and exiting from TrustZone. Next, we implement a program with TrustZone entry/exit instructions, as well as protected running steps. We then test it with controlling of the frequency of entry/exit instructions. We measure the FLUSH operation overhead according to different frequencies, and discuss the defense using FLUSH when exiting from TrustZone.

3.4 Threat Model and Assumptions

In this paper, we assume that the operating system is not compromised so that the attackers are forced to use covert channels or side channels without explicitly violating access control policies enforced by the operating system or other protection mechanisms. We assume that the attacker has sufficient privilege to access the memory access time. This is also needed for the covert channel, and for the performance analysis of the covert channel.

4 Implementations

4.1 Process Structure on Cortex-A Platform

As mentioned above, the very first step for our experiment is to calculate the cost of entering and exiting from the TrustZone. On ARM Cortex-A Platform, an instruction smc is used for connecting the secure world and non-secure world. While in normal non-secure world, some code could call privileged smc instruction. Then, secure world monitor will be triggered after validation. After execution of secure code, the return of the execution also calls smc to get back to the normal world. There are many open-source test platform to measure the world switch latency, and in this experiment, we use the well-known QEMU to test. It had been developed since the first patch published in 2011, and been patched by many manufacturers including Samsung, utilizing ARM TrustZone for security design.

The process structure is show at Figure 1. When there are smc instructions trigger the TrustZone entry/exit at step 1, we trap the instructions and start using perf and other time measurement tools to calculate clock cycles they take to finish switching between trust environment and outside memory at step 4. We also FLUSH cache every time when we exit from TrustZone and see the difference in performance overhead by different frequency of TrustZone related instructions. Basically the time measurements work in steps 2-4, and we have discussions based on the measurements.

4.2 Process Structure on Cortex-M Platform

On ARM-v8 platform, SG/BXNS instructions are used to enter and exit from TrustZone. As there were almost no proper TEEs for ARMv8-M on the market as we were testing, we use a testing program instead. SG (Secure Gate) instruction is called by non-secure world code that wants to trigger TrustZone protection. Unlike Cortex-A structure, on ARMv8-M, the page table is not used, so the memory is fully mapped with different regions. When SG instruction is called, the reserved regions for secure world are used to execute the protected part of the code. After the secure execution within TrustZone, the code has an exit called BXNS/BLXNS (Back to Non-Secure) that can lead the execution to other region besides protected ones by TrustZone. We make use of the mechanism of this, and the structure of the testing program is as Figure 2 shows.

Similar to process structure on cortex-A, we calculate time stamps in steps 2-4. As there are no perf-like instructions as of cortex-A platform, we have to consult special registers to retrieve information about the time stamps. We consult registers like $TPIU\_ACPR$, and TPIU Asynchronous Clock pre-scaler Registers to set up time measurements.

The term 'cache' here on ARMv8-M is part of normal memory being set as 'cacheable'. In other words, it is a region set aside for possible cache using. On Cortex-A series chips or x86 chips, cache flush operations are just some instructions with privileges. However, the case are different on ARMv8-M. The allocation of a memory address to a cache address is defined by the designers of the applications. Because of the special structure of ARMv8-M, the cache FLUSH operations are sets of DSB (Data Synchronization Barrier) operations, with address-related instructions.

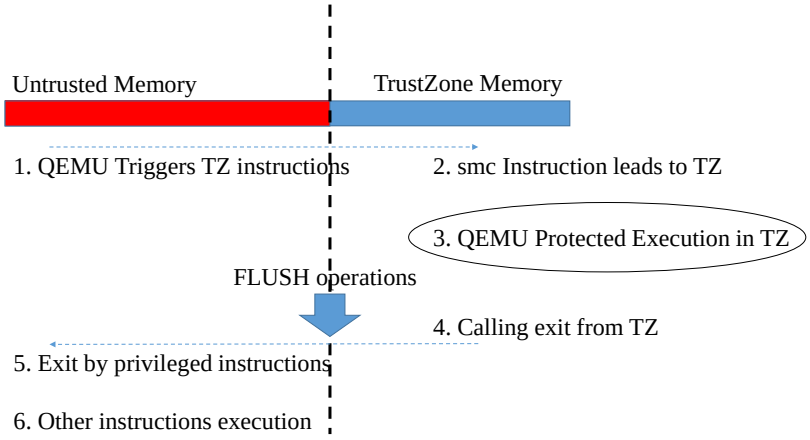


Figure 1: Process Structure on Cortex-A

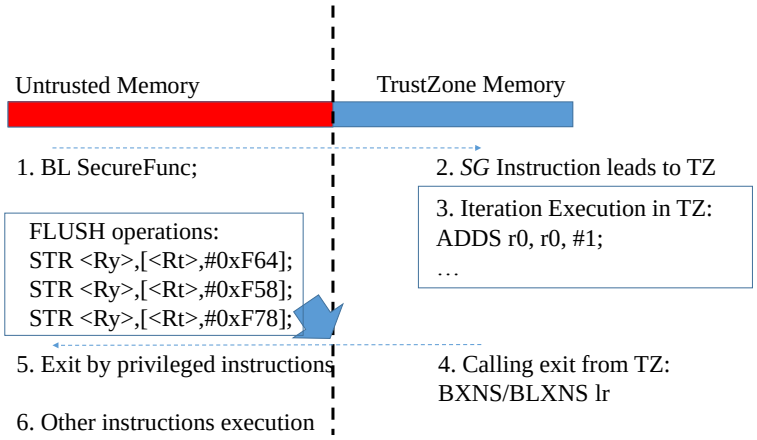


Figure 2: Process Structure on Cortex-M

4.2.1 Injecting FLUSH Operations on ARM Platform

We use cache FLUSH operations as injected noise into cache, giving interference to attacker's time measurement phase. On ARM platform, we consider using assembly code for noise injection, which is quite different from x86 platform on which we have straightforward $clflush$ operations to do so. As a result, we have to take a look at the cache allocation on ARM, and use inline assembly codes to implement cache FLUSH operations.

It is usual to clean the cache before flushing it, so the external memory is updated with any dirty data. First, we initialize inner loop and outer loop to build the cache FLUSH operations line by line. Then, we generate segment address. After that, we call the instruction MCR to specified registers with parameters based on the cache size. The following segments show the inner loop and FLUSH operations. For outer loop, we just move the address to the next line each time in the iteration, which is easy to implement.

```
MOV r0, #0 ; //Clear R0;
ORR r2, r1, r0 ; //Generate segment and line address
MCR p15, 0, r15, c7, c10, 3 ;
//Flush DCache;
ADD r0, r0, #0x20 //To the next cache line;
```

On ARMv7 or higher, the cache FLUSH operations that are privileged and can be handled by ARM. In the code above, we can see the assembling code of flushing the cache uses `MCR` (Move to Coprocessor from Registers). The privileged operation using this can be trapped by the system, and system handles the operation referring to it. The reason is that, when an instruction uses `MCR` or `MRC`, the registers CP14 and CP15 are taken access. These registers are designed by ARM with special purpose, and used only for cache maintenance. For ARM, it has a system call which takes an array of those operations each specified by the struct called *mmuext_op*. This call allows access to various operations which must be performed with privileged level, like TLB operations, cache operations, and loading descriptor table base addresses.

4.2.2 Side-Channel Attack Implementations

In this paper, we also have implemented test cases of side-channel attack on ARM platform, targeting to *libjpeg* and some image file. In real life, shared libraries like *libjpeg* are always the target to be attacked by malicious users, since they can be utilized to retrieve image which contains information that is easy to read. On the other hand, when we try to use noise injections into the message channel, the attackers get low quality images with noises, but they can still retrieve enough information from the picture if it's still readable. Sometimes they can even use Error Correction Code (ECC) to try fixing their results. ECC code can fix images with low level noise injection, but perform worse when we inject too much noise that exceeds the threshold based on the signal/noise ratio.

The side-channel attack we implemented are recurrence of the experiments in the paper by Ruby Lee and his team [12]. We ported to ARM platform and also inject noise into the process of this attack. Our attack tests are based on Flush+Reload attack, and the results are shown and discussed in Evaluation chapter.

The following figure shows the structure of a side-channel attack 3. The side-channel implementations are based on FLUSH+RELOAD type attack. The victim process is randomly accessing cache lines, and the attacker FLUSH cache in a certain frequency which is also its attack frequency. Then, it reloads the cache lines and measure the access times. By comparing the access time difference, the attacker can infer which cache line is being accessed by the victim.

As shown in the structure figure, time measurement is the key to the success of attacker launching cache-based side-channel attack. Unlike performance counters on x86, on ARMv8 platform, there are no instructions like perf to collect time-related performance counters from the system layer. Another challenge is that we cannot use *rdtsc* instruction to get time stamps as we often do on x86. Additionally, some other coarse-grained way like gettimeofday() certainly does not work. Given these limitations, we have to be back to hardware, and look at ARM structure itself.

We look up ARM whitebook and find some registers that we can retrieve time stamp information. However, when consulting with these registers, we have to enable them from kernel mode. By default, the access to these registers are disabled. The following code segment shows the instructions for calculating time:

```
ISB; MRS%0, cntvct_el0 ;
//process execution ;
```

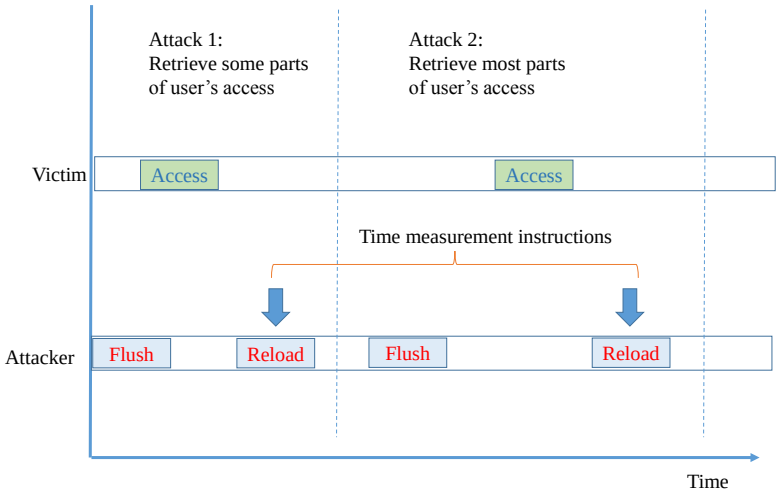


Figure 3: Side-Channel Attack

Table 2: TrustZone-Related Instruction Cost on Cortex-A

Tests	Direction	Average cost (Clock Cycles)	Time on 800Mhz
P0_nonsecure_check_register_access	Non-secure to Secure	1950	2.43us
P0_secure_check_register_access	Secure to Non-secure	2200	2.75us

```
ISB; MRS%0, cntvct_el0 ;
ISB; MRS%0, cntfrq_el0 ;
```

We store the time stamps in two arrays and calculate the time based on these raw data. The instructions are privileged, and we can use time stamps for many monitor jobs. *cntfrq_el*0 is used for reading current running frequency, which is not always the CPU frequency or clock frequency.

5 Evaluation

In this section, we introduce our experimental results and discussions, both on ARM Cortex-A and Cortex-M platforms.

5.1 Experimental Results

5.1.1 Cost of Entering and Exiting from TrustZone on Cortex-A

QEMU with ARM TrustZone provides us a variety of tests. The tests behave as we users initiating secure operations from user mode. The test functions validate the TrustZone features of QEMU, and utilizing the features of the functions themselves. We have tests on read/write from non-secure world to secure world and vice versa. The results are shown as Table 2 shows.

5.1.2 Percentage of TrustZone-Related Instructions

We write a script based on the above write/read code. In the script, there is a loop called in and runs several times as a workload. We use Ubuntu 16.10 as the normal world OS, with 26 processes running

Table 3: Different Categories of TrustZone-Related Instructions

Type	Percentage
Non-secure to Secure Test R/W	2.87%
Secure to Non-secure Test R/W	2.91%
Others (Access from Background)	0.01%

on background, including the workload we use for testing. We count the smc-related instructions that belongs to TrustZone-related operations, and analyze the attributions of them. According to our test, the instructions takes up less than 6% of the total instructions running, with these three different categories as shown on Table 3.

In normal using conditions, however, the manufacturers are not using TrustZone that often. Thus, the test here can be the upper bound or 'worst case' of the utilization of TrustZone-Related instructions. Normally, the non-secure world does not have to call in the secure world too often.

5.1.3 Performance Overhead by FLUSH Operations

It is already known that ARM TrustZone on Cortex-A series are not going to clean the cache when exiting from the secure world to non-secure world. As a result, there are possibilities for the attackers to make the most of the last level cache and conduct cache-based attacks. For example, the side-channel attack of FLUSH+RELOAD, PRIME+PROBE are both found practical on the environment with TrustZone on ARM Cortex-A, some even with a fiercely high bandwidth. On the other hand, if we can FLUSH the cache every time on the 'exit' to the normal non-secure world, then it can be expected that the bandwidth of the side-channel attack can be limited to a number that is worthless to the attackers to gather the information possibly leaked by the smc operations.

We still test the performance using our test model. In this test, we are adding cache FLUSH operations on every smc instruction that calling exit from the secure world to non-secure world. On that situation, we measure the performance overhead by comparing the clock cycles of execution. At the same time, we change the percentage of TrustZone-related instructions to see the difference in the overhead. The results are shown on Figure 4 and Figure 5.

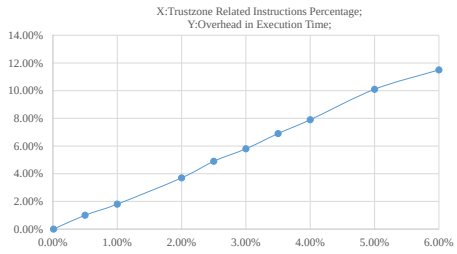


Figure 4: TrustZone Related Instructions and Their Overhead

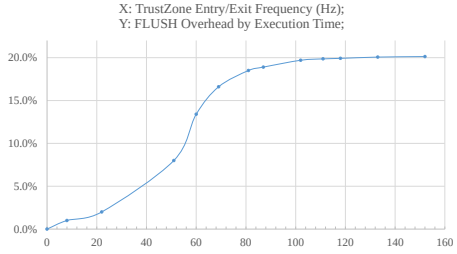


Figure 5: TrustZone Entry/Exit Frequency and FLUSH Overhead

Figure 6: Noise Injection on ARM

Table 4: TrustZone-Related Instructions Cost on ARMv8-M

Operation	Direction	Cost on Average (Clock Cycles)
SG	Non-Secure to Secure	3.5
BXNS/BLXNS	Secure to Non-Secure	5.2

5.1.4 Side-Channel Experiments on Cortex-A

On Figure 5, the results show the impact with the system handling with cache FLUSH operations. We change the percentage of FLUSH instructions, and receive different level of the channel performance and bandwidth. Here we are using in-line assembly code to record time in order to receive relatively accurate running time for each test case. From the results, we can see the increasing overhead with the increase of FLUSH instructions. With less cache FLUSH instructions added into the system, the bandwidth of the covert channel or side-channel is increasing quickly. We may inject less FLUSH operations for performance concerns, but we cannot do this in real tests, since at this situation the system is losing control of the cache and memory access, leaving a highly risky flaw to the attackers.

We also use image files for the test. At that time, the noise in the picture increases with more percentage of FLUSH operations injected into the system. Figure 6 shows the difference. This is the case when we set up 50Hz FLUSH operations to be added, while another process sending information by the transmission of an image file with 201kb size. We use side-channel strategy to try retrieving the image, in unlimited time (which is not the real-life case since the attackers must act quickly to launch their attacks). We give the attacher unlimited time just to show how serious and practical a side-channel threat can be.

Based on the figure above and experimental results, we can see that with FLUSH operations added into the system, the attacker is with more difficulty in retrieving information in the system. Even with little overhead, the defense can be effective. However, the attacker can still have parts of the information, which is another problem of balancing in performance and effectiveness. We have discussions of that in our other papers, introducing adaptive defense design based on balance of these parameters.

5.1.5 Experimental Results on ARMv8-M

According to our experiments, the testing case triggering TrustZone operations SG and BXNS. As every region is fixed in the memory, the costs of entering and exiting from TrustZone are surprisingly much lower than ARM Cortex-A series chips. The results are shown at Table 4.

We measure the performance of the FLUSH operations using our testing program shown at Figure 2. We add FLUSH operations before executing BXNS/BLXNS operations to ensure there is nothing left

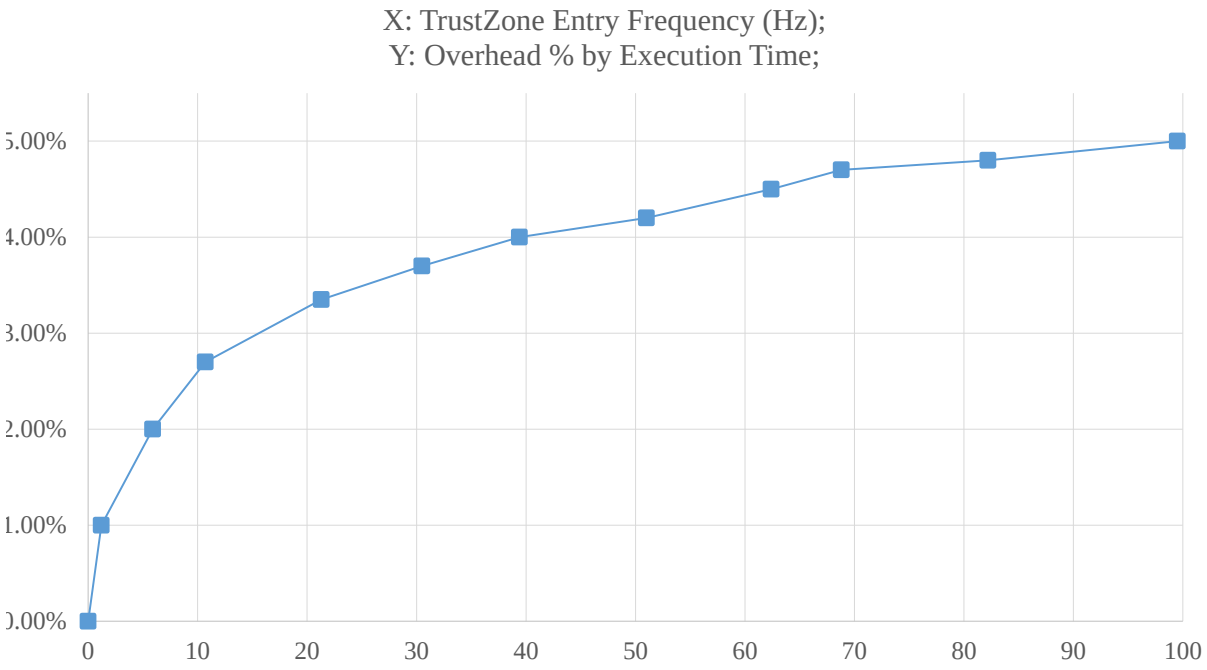


Figure 7: TrustZone Entry/Exit Frequency and FLUSH Overhead on ARMv8-M

when exiting from TrustZone. We measure the overhead by the FLUSH operations, and we also change the outer loop to have different frequencies of TrustZone entries and exits. The results are shown at Figure 7.

5.2 Discussions

5.2.1 TrustZone Usage Frequency and Flush Overhead

According to our experimental results, on ARMv8 platform, the system is connecting with TrustZone with very low frequency, taking up less than 10% of the instructions at most. Some specific instructions trigger the secure gate of TrustZone. However, when the contexts running in secured memory finish, TrustZone does not clean the cache before exit, leaving some risks here. Based on low frequency and overhead from TrustZone related instructions, we can FLUSH the cache every time when exiting from TrustZone, and still keep a low overhead of less than 20% on Cortex-A chips. This design will let the system manufacturer to put protected or private contexts into TrustZone and with no worries about side-channel attack when exiting from it.

5.2.2 TrustZone Discussion on Cortex-M

Unlike Cortex-A series, ARMv8-M based on Cortex-M structure is designed to have low energy cost and with much simpler system, which is thought to fit for mobile or home devices. At this case, the performance overhead brought by security protection should be controlled in a very low number. According to our experimental results, on Cortex-M structure, the secure gate instructions take much less clock cycles to execute, making it a good choice on the basis of security design. When we add FLUSH operations on exit instructions, we have even lower overhead comparing with Cortex-A chips, having less than 10% overhead at most. It is a practical design for the manufacturer to introduce and not hard to develop. On

the other hand, they could put protected data and instructions into the secure enclave of TrustZone.

5.2.3 Cache Based Defense on ARM Platform

Though we have no perfect way to take the place of validating cache and cleaning the TLB entries, we still have some idea for possible solutions, because there are some potential for speeding up and getting better performance. For example, we can move the FLUSH operations out from the privileged level, and try implementing another framework to ensure the security of this type of operations, while maintaining low overhead. In this paper, we quantitatively discuss the security design for dealing with FLUSH operation requests, and there are still some more topics to research on.

6 Conclusion

In this paper, we have some discussion on the effectiveness and cost of attack and defense based on ARM platform. We start from investigating the cache-based attacks. Then we design and implement some tests on ARM platform, both on ARM Cortex-A and ARMv8-M series chips. It is shown that the side-channel attack and other types of exploitations are practical and serious, causing loss to users' privacy and security. From our experimental results, TrustZone can be utilized to help defending against side-channel and covert channel attacks, but it must have an adaptive ways to manage cache operations. On the other hand, it is practical to implement FLUSH based defense on ARM platform, with reasonable overhead and good effectiveness.

In the future, we need to develop some defense framework on ARM platform, based on FLUSH operations and secure gate entry/exit instructions. The challenge will be the difference in structures of ARMv8 platform, and real-life limitations like power consumption, portable needs and other challenges. However, it is promising that ARM platform can provide the users with an environment in balance of performance, privacy, security and good mobility as well.

Acknowledgements

This paper and research project are sponsored by NSF CREST Grant HRD-1736209 and NSF Grant 1634441. The grants are for security research on cloud and systems. This research is performed in the Institute for Cyber Security (ICS) lab in University of Texas at San Antonio, and Computer Science Department in Roosevelt University.

References

[1] J. Amacher and V. Schiavoni. On the performance of arm trustzone. In *Proc. of the 19th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS'19), Kongens Lyngby, Denmark*, volume 11534 of *Lecture Notes in Computer Science*, pages 133–151. Springer, Cham, June 2019.

[2] D. J. Bernstein. Cache-timing attacks on AES. Technical report, 2005.

[3] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert. Software mitigations to hedge aes against cache-based software side channel vulnerabilities. `https://eprint.iacr.org/2006/052` [Online; accessed on December 15, 2020], 2006. Cryptology ePrint Archive: Report 2006/052.

[4] J. V. Cleemput, B. Coppens, and B. De Sutter. Compiler mitigations for time attacks on modern x86 processors. *ACM Transactions on Architecture and Code Optimization*, 8(4):23:1–23:20, January 2012.

[5] S. Crane, A. Homescu, S. Brunthaler, P. Larsen, and M. Franz. Thwarting cache side-channel attacks through dynamic software diversity. In *Proc. of the 22nd Annual Network and Distributed System Security Symposium (NDSS'15), San Diego, California, USA*. Internet Society, February 2015.

[6] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger. Trustshadow: Secure execution of unmodified applications with arm trustzone. In *Proc. of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys'17), Niagara Falls, New York, USA*, pages 488–501. ACM, June 2017.

[7] B. Gülmezoğlu, M. S. İnci, G. Irazoqui, T. Eisenbarth, and B. Sunar. A faster and more realistic flush+reload attack on aes. In *Proc. of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE'15), Berlin, Germany*, volume 9064 of *Lecture Notes in Copmputer Science*, pages 111–126. Springer, April 2015.

[8] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan. vtz: Virtualizing arm trustzone. In *Proc. of the 26th USENIX Conference on Security Symposium (SEC'17), Vancouver, British Columbia, Canada*, pages 541–556. USENIX Association, August 2017.

[9] G. Irazoqui, T. Eisenbarth, and B. Sunar. S$a: A shared cache attack that works across cores and defies vm sandboxing – and its application to aes. In *Proc. of the 2015 IEEE Symposium on Security and Privacy (S&P'15), San Jose, California, USA*, pages 591–604. IEEE, May 2015.

[10] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar. *Wait a Minute! A fast, Cross-VM Attack on AES*, volume 8688 of *Lecture Notes in Computer Science*, pages 299–319. Springer, 2014.

[11] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanović. Keystone: A framework for architecting tees. arXiv:1907.10119, July 2019. `https://arxiv.org/abs/1907.10119` [Online; accessed on December 15, 2020].

[12] F. Liu and R. B. Lee. Random fill cache architecture. In *Proc. of the 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, UK*, pages 203–215. IEEE, December 2014.

[13] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *Proc. of the 2015 IEEE Symposium on Security and Privacy (S&P'15), San Jose, California, USA*, pages 605–622. IEEE, May 2015.

[14] N. Liu, M. Yu, W. Zang, and R. Sandhu. On the cost-effectiveness of trustzone defense on arm platform. In *Proc. of the 21st International Conference on Information Security Applications (WISA'20), Jeju Island, South Korea*, volume 12583 of *Lecture Notes in Computer Science*, pages 203–214. Springer, August 2020.

[15] D. Page. Partitioned cache architecture as a side-channel defence mechanism. `http://eprint.iacr.org/2005/280` [Online; accessed on December 15, 2020], 2005.

[16] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral. Ltzvisor: Trustzone is the key. In *Proc. of the 29th Euromicro Conference on Real-Time Systems (ECRTS'17), Dubrovnik, Croatia*, volume 76. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[17] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proc. of the 2017 Annual Network and Distributed System Security Symposium (NDSS'17), San Diego, California, USA*. NDSS Symposium, February 2017.

[18] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *Proc. of the 34th Annual International Symposium on Computer Architecture (ISCA'07), San Diego, California, USA*, pages 494–505. ACM, June 2007.

[19] Z. Wang and R. B. Lee. A novel cache architecture with enhanced performance and security. In *Proc. of the 41st IEEE/ACM International Symposium on Microarchitecture (MICRO'08), Lake Como, Italy*, pages 83–93. IEEE, November 2008.

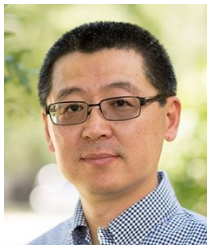
[20] Z. Wu, Z. Xu, and H. Wang. Whispers in the hyper-space: High-bandwidth and reliable covert channel attacks inside the cloud. *IEEE/ACM Transactions on Networking*, 23(2):603–614, April 2015.

[21] Y. Yarom and K. Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *Proc. of the 23rd USENIX Conference on Security Symposium (SEC'14), San Diego, California, USA*, pages 719–732. USENIX Association, August 2014.

[22] Y. Zhang, Y. Chen, C. Bao, L. Xia, L. Zhen, Y. Lu, T. Wei, and X. Baidu. Adaptive kernel live patching: An open collaborative effort to ameliorate android n-day root exploits. Black Hat USA 2016, 2016. https://www.blackhat.com/docs/us-16/materials/us-16-Zhang-Adaptive-Kernel-Live-Patching-An-Open-Collaborative-Effort-To-Ameliorate-Android-N-Day-Root-Exploits-wp.pdf [Online; accessed on December 15, 2020].

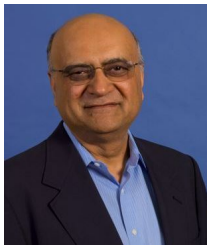
Author Biography

Naiwei Liu is currently a Ph.D student graduating in December 2020. His research is about cyber security and system security, focusing on security design on different hardware platforms. His research work have been published in multiple publications, and his doctoral dissertation is about ARM security design, guided by Dr. Meng Yu and Dr. Ravi Sandhu.



Meng Yu works in Roosevelt University as Professor and Department Chair since 2018. He had been a postdoctoral research associate in Pennsylvania State University, at Prof. Peng Liu's Cyber Security Lab. He has worked as faculty for 16 years in 5 different universities in the U.S., adviced Naiwei Liu's Ph.D studies and research.

Wanyu Zang works in Roosevelt University as Lecturer since 2018. She received her Ph.D degree in Computer Science in 2001 from Nanjing University, China. Her research interests are in the fields of security, especially in network security and cloud security.



Ravi Sandhu is a Professor and leader in Institute for Cyber Security Lab (ICS Lab) in University of Texas at San Antonio. He has been working for many decades on security research. His research fields are related to cyber security and with 300+ papers published, he is one of the leading researchers in cyber security area.