SH3ARS: Privilege Reduction for ARMv8.0-A Secure Monitors

Jonas Röckl*, Julian Funk*, Matti Schulze*, Tilo Müller†
*Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
{jonas.roeckl, julian.funk, matti.schulze}@fau.de
†Hof University of Applied Sciences, Germany
tilo.mueller@hof-university.de

***Abstract*—The ARM TrustZone Trusted Execution Environment (TEE) allows software to run in an isolated environment, separated from the untrusted OS. The isolation is based on the Secure Monitor (SM), software running at the most privileged hardware level, with unrestricted access to all system resources, including those of the TEE. Critically, recent research revealed widespread vulnerabilities in SMs that break the TEE isolation and, thus, undermine the very purpose of the TEE.**

To this end, we present SH3ARS, a fundamental restructuration of the SM firmware that reduces the privileges of the SM and restores ARM TrustZone isolation. SH3ARS modifies the SM to irrevocably relinquish access to memory outside its own address space through a *page table latching* mechanism. Furthermore, we introduce *guards*, carefully crafted, gadget-free code sequences, that supervise the context switch to and from the TEE, preventing code-reuse attacks against the TEE – a technique we refer to as *SMC-oriented programming*. Relying on software changes, SH3ARS ensures TEE isolation guarantees, even if the SM is compromised.

We apply SH3ARS to the reference implementation of the SM on ARMv8.0-A, as deployed on millions of devices. We implement a proof of concept on real hardware, and our evaluation shows that the overhead is lower than 6% for most workloads.

***Index Terms*—Trusted Execution Environments, Firmware**

I. Introduction

The ARMv8-A architecture has emerged as the leading platform for mobile computing [1], IoT [2], embedded systems [3], and edge computing [4], while also gaining a growing market share in personal computing [5] as well as cloud computing [6]. Hence, it is no surprise that ARM-based devices have become a prominent target for attackers [7]–[10].

Due to their large code base, general-purpose OSs are likely to contain vulnerabilities, posing a security risk [11]–[13]. The ARM TrustZone Trusted Execution Environment (TEE), which is widely available on ARMv8-A systems [14], is designed to protect security-critical functionality (e.g., fingerprint authentication [15], disk encryption [16], [17], and electronic payment [18]) from potential OS-level vulnerabilities. While regular applications run in the Non-Secure World (NW) based on the general-purpose OS, security-critical Trusted Applications (TAs) run in the so-called Secure World (SW), based on a small-scale special purpose OS referred to as TrustZone Operating System (TZOS). This ensures that data and code of TAs remain isolated, even if the NW OS is compromised.

The isolation of TrustZone is based on a trusted software component called the Secure Monitor (SM), running at the most privileged hardware level, with unrestricted access to system resources (Figure 1). Most devices rely on Trusted Firmware-A (TF-A) [19], the official reference implementation for SM firmware. Besides managing context switches between the NW and SW, the SM also provides runtime services (e.g., power level management [20]) to less-privileged system software like the NW OS. The Secure Monitor Call (SMC) instruction is used to invoke runtime services and initiate context switches between the NW and the SW. When the NW OS or the TZOS executes an SMC, the system switches to the SM, which processes the untrusted inputs from the less-privileged software and handles the request.

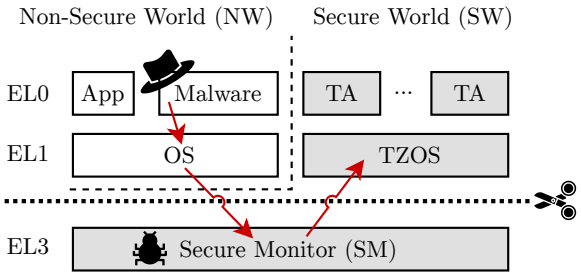

Fig. 1. Overview of the ARMv8.0-A architecture. An attacker in the untrusted NW (black hat) exploits vulnerabilities (bug) in SM firmware. Upon taking over the SM, the attacker gains architectural access to all memory, including the one of the SW. Metaphorically, SH3ARS cuts off (scissors) the privileges from the SM. EL2, which can host a hypervisor, is left out for clarity.

A security vulnerability within the SM firmware undermines the very purpose of the TEE. By exploiting vulnerabilities in the SM firmware such as memory corruption bugs, attackers that compromised the NW can escalate their privileges to the SM. Since the SM is typically omnipotent, the attacker gains full control over the system, including access to the TZOS, and the TAs, subverting all hardware-based isolation layers.

Recent research shows that this is not merely a theoretical attack but a practical threat. Cerdeira et al. present a security analysis of TrustZone-based systems, including a validation bug in the SM that can lead to a full system compromise [2]. Moreover, through fuzzing of SM binaries, Lindenmeier et al. identified 34 bugs and 17 critical software security vulnerabilities in SM firmware from six popular vendors, namely Samsung, Huawei, Intel, NXP, Xilinx, and Nvidia [1].

We observe that the root cause of the severe consequences stemming from vulnerabilities in SM firmware – such as out-of-bounds reads (e.g., CVE-2022-47630, CVE-2023-31339), out-of-bounds writes (e.g., CVE-2016-10319, CVE-2023-49614), and local privilege escalation or information disclosure flaws (e.g., CVE-2022-38787, CVE-2023-22327) – lies in the omnipotent privileges granted to SM firmware. Although formal verification can help prove the absence of certain vulnerabilities in system software [21]–[26], its application remains challenging. Therefore, we call for pragmatic solutions for protecting the device's most valuable secrets, even in the presence of vulnerabilities in SM firmware.

A. Contributions

We propose SH3ARS (**S**oftware **H**ardening for EL**3** on **AR**Mv8.0-A **S**ystems), a fundamental restructuration of the SM firmware on TrustZone-based systems that ensures that the TZOS and TAs remain isolated, even if the SM is compromised. Metaphorically, SH3ARS cuts off the privileges from the SM (Figure 1) and is, to the best of our knowledge, the first design to reduce the privileges of ARMv8.0-A SMs.

Based on the observation that SM firmware does not require altering its memory mappings after the initial setup, we propose *page table latching* as a novel technique for SM firmware. This technique ensures that the SM firmware irrevocably drops the privileges to access any memory region beyond its own and securely sets up memory access attributes *before* potentially malicious input from the NW is processed. Consequently, the SM firmware isolates itself from memory regions associated with the TZOS and the TAs.

However, memory protection mechanisms alone are insufficient. To manage context switches between the NW OS the and TZOS, the SM saves and restores the respective CPU contexts for lower privileged hardware levels. By tampering with the saved CPU context of the TZOS, which is stored in SM memory, a compromised SM can launch attacks similar to Return-Oriented Programming (ROP) [27] and Jump-Oriented Programming (JOP) [28] against the TZOS. We show how an attacker can chain existing gadgets in the TZOS and refer to this technique as *SMC-oriented programming*.

Therefore, we also need to ensure context protection. We propose and implement strategies for verifying the integrity of the saved TZOS context prior to switching to the TZOS. Since we assume that the SM firmware is vulnerable, it is crucial to ensure that an attacker cannot trick the SM into bypassing the verification checks. Although the fixed memory mappings prevent injecting malicious executable code into the SM, code-reuse attacks *on the* SM would still allow evading the context verification. For this reason, we introduce *entry guards* and *exit guards* – sequences of instructions that are guaranteed to be executed upon switching to the SM and before returning to the TZOS, even in the presence of code-reuse attacks.

We focus on ARMv8.0-A hardware as it is deployed on millions of real devices [29], [30]. Despite this, the ARMv8.0-A architecture suffers from a design flaw. Specifically, the TZOS has unrestricted access to memory on ARMv8.0-A, including that of the SM. Therefore, we describe how current approaches for state-of-the-art TZOS privilege reduction [31] can be integrated into SH3ARS, enabling *mutual* isolation between the SM and the TZOS on ARMv8.0-A hardware.

We prototype SH3ARS on real hardware, specifically the i.MX 8MQuad Evaluation Kit, which features an NXP i.MX8M Cortex-A53 CPU and a Cortex-M4 coprocessor. The device runs a vendor-specific fork of TF-A as its SM firmware and OP-TEE [32], an open-source TZOS. We show that the increased isolation does not come at the cost of the functionality of a typical TEE software stack. SH3ARS relies exclusively on standard ARMv8.0-A features, making it compatible with a broad range of existing devices. In summary, we make the following contributions:

- We propose and implement *SH3ARS* to deprivilege otherwise omnipotent SM firmware on ARMv8.0-A-based systems. Relying on source code modifications and binary analysis, SH3ARS protects the TZOS and TAs, even if the SM firmware is compromised, thereby restoring the isolation of the ARM TrustZone TEE.
- We show that SH3ARS only increases the Trusted Computing Base (TCB) by 0.53% and discuss security implications. Our evaluation shows that the performance overhead is lower than 6% for most workloads.
- We release SH3ARS' source code as open-source.[1]

B. Outline

This publication is structured as follows. We first give background knowledge on ARMv8.0-A, before we specify our threat model (Section III). In Section IV, we explain how SH3ARS prevents the SM firmware from accessing TZOS and TA memory. Section V describes how the saved TZOS context is protected from tampering. We discuss security implications (Section VI) and conduct a performance evaluation of SH3ARS (Section VII). In Section VIII, we describe how to integrate a state-of-the-art technique to deprivilege the TZOS. We deal with further related work in Section IX and outline limitations and future work in Section X.

II. Background

A. ARMv8.0-A Architecture

The ARMv8.0-A architecture defines four privilege levels, which are referred to as Exception Levels (ELs), ranging from the highest-privileged EL3 to the lowest-privileged EL0 (Figure 1). EL0 hosts user-mode applications, which are managed by an OS at EL1. EL2 is designated for hardware-assisted virtualization. Since hypervisors are still untrusted, this does not influence the design of SH3ARS. Therefore, we omit EL2 to maintain simplicity. After a reset, boot loaders [31], [33] run before passing control to the SM firmware at EL3, which bootstraps the rest of the system. At EL3, the CPU is always in the SW state. The other ELs are available in both NW and SW states. Their SW counterparts are referred to as S-EL0 and S-EL1.

[1] https://github.com/SH3ARS/SH3ARS

The NW interacts with the SW in two ways. First, the NW can request runtime services from the SM (Section I). Second, a user-level application in the NW can request the execution of a TA. In either case, the NW OS issues an SMC. The control flow changes to the SM, which saves the NW's CPU context in memory. A request to a runtime service is handled directly by the SM. If the NW requests a TA, the SM restores the saved CPU context of the TZOS from memory to the system registers and initiates a switch from EL3 to the TZOS at S-EL1. The TZOS scheduler dispatches the request to forward it to the corresponding TA on S-EL0.

ARMv8.0-A system registers are named *REG_ELx*, where *ELx* represents the EL for which the register takes effect. For example, *ELR_EL3* indicates the instruction pointer of a lower-level EL to return to after leaving EL3. Higher-privileged ELs can access the system registers of lower-privileged EL.

III. Threat Model

We focus on ARMv8.0-A-based devices equipped with the ARM TrustZone TEE and assume that the hardware works according to the manufacturer's specifications. Consistent with TrustZone's standard threat model, we assume that the NW is under the control of an attacker. We assume that, ultimately, the attacker's goal is to access or manipulate data from the TZOS or the TAs, which hold the device's secrets.

We require that the integrity of the bootloader and the SM binary is verified during the boot process through secure boot mechanisms, as integrated into most ARMv8.0-A platforms [31]. Therefore, we assume that the SM firmware as well as all prior bootloaders are *initially*, i.e., before processing any untrusted input from the NW [34], benign.

We assume that the SM firmware includes undisclosed security vulnerabilities, as undermined by recent research [1], [2] and vulnerabilities (e.g., CVE-2022-47630, CVE-2023-31339, CVE-2016-10319, CVE-2023-49614, CVE-2022-38787, and CVE-2023-22327). Thus, our threat model is stronger than those typically considered in TrustZone-based systems, which assume that the SM is free of vulnerabilities.

We assume that the attacker relies on the runtime interface between the NW and the SW, i.e., SMCs, to feed malicious inputs into the SM firmware, the TZOS, or the TAs. Exploiting a vulnerability in the SM (Figure 1), the attacker aims for accessing confidential data of TZOS and TAs.

We exclude denial-of-service attacks against the TEE. Further, physical attacks and side-channel attacks are out of scope.

IV. Memory Protection

We first explain how SH3ARS ensures that the SM firmware is prevented from accessing TZOS and TA memory.

A. Isolation in Time

We observe that TF-A does not require altering its memory mappings after being set up. Therefore, we propose modifications to the SM firmware that permanently revoke the privilege to access arbitrary memory after the SM's initialization. After dropping the privileges, full memory access can no longer be regained, until the system is reset, thus isolating the SM from the rest of the SW.

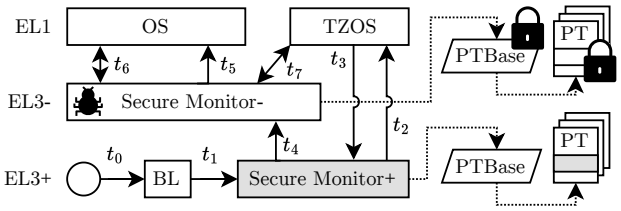

Fig. 2. SH3ARS' boot flow. At power-on (t_0), one or more bootloaders (BL) execute, passing control to the SM (t_1). Typically, the SM sets up the MMU by creating page tables (PT) and writing a configuration register (PTBase) to point to them. Afterward, the TZOS (t_2 and t_3) initializes. With SH3ARS, a privilege drop occurs (t_4), locking system registers (Section IV-B) and page tables (Section IV-C) until the next reboot (padlocks). Finally, the NW OS (t_5) boots and, later on, requests services from the SM (t_6), which might lead to interaction with the TZOS (t_7).

Figure 2 shows the boot flow of a SH3ARS-enabled device. Let t_0 denote the point in time when the device is powered on and the bootloader is executed. t_1 marks the moment when the SM executes for the first time to initialize the platform. We refer to EL3 which has unrestricted access to the system's memory as Privileged EL3 (EL3+). At t_2, the SM transfers control to the TZOS for its initialization, which is finished at t_3. At t_4, SH3ARS drops its privilege to access arbitrary memory. We refer to this execution state as Deprivileged EL3 (EL3-). Consequently, we divide the SM firmware into two parts logically, referred to as SM+ and SM-, running at EL3+ and EL3-, respectively. t_5 denotes the initialization of the NW OS, which continues execution thereafter. During the runtime, the NW OS requests services either from the SM (t_6) or from TAs. In the latter case, the SM forwards the request to the TZOS (t_7), which executes the TA and returns the result.

It is important to note that the SM can only be exposed to untrusted user input from the NW after the NW has been initialized (t_5). With SH3ARS, SM+ is never exposed to untrusted input that could potentially exploit vulnerabilities. SM- may be exploited. However, SM- lacks the privileges to access memory beyond its own address space, which does not overlap with TZOS or TA memory.

B. Memory-Critical System Registers

The ARMv8.0-A memory management system is configured through system registers and page tables [35, p. D8-6448]. In this section, we explain how SH3ARS prevents SM- from changing the registers related to the memory configuration.

Definition. *We define a system register as memory-critical if it affects the memory access permissions of EL3.*

We identify five memory-critical registers: *SCTLR_EL3* holds a bit for activating the MMU, *TCR_EL3* configures virtual address translation, and *TTBR0_EL3* holds the base address of the page tables. Finally, *MAIR_EL3* and *AMAIR_EL3* store memory attributes encodings.

a) Identifying System Register Access: Given the SM firmware, we identify all write instructions targeting memory-critical registers. On ARMv8.0-A, configuration registers can only be written with the `MSR` instruction and read with the `MRS` instruction [35, p. C5-808]. These instructions copy the value in a general-purpose register to a system register and vice versa. We observe that TF-A, ARM's reference implementation of SM firmware, only accesses memory-critical registers when bootstrapping the system (t_1–t_3, Figure 2). They are not required to be accessed when handling runtime requests to the SM, the TZOS, or one of its TAs (t_5–t_7, Figure 2).

b) Code Splitting: We split the code of the SM firmware into SM+ and SM-. We extract the symbols (C functions) that write to memory-critical registers from the disassembled machine code. Next, we modify the linker script of the SM to place these symbols in a dedicated code segment. During the execution of SM+ at EL3+, this segment is executable. This approach allows access to the registers during boot time without requiring extensive modifications to the SM firmware.

When switching from SM+ to SM- (t_4, Figure 2), the segment with the instructions to memory-critical registers is configured as *non*-executable. Since ARMv8.0-A enforces instruction alignment, with each instruction being exactly four bytes wide, we can trivially verify with static analysis that none of the instructions contains the opcodes for `MSR` or `MRS` instructions to memory-critical registers [36]–[38]. Consequently, with the current memory mapping, no instruction that accesses a memory-critical register is executable in SM-.

C. Page Table Latching

An SM- attacker could alter the page tables [35, p. D8-6486] to re-map the segment with the instructions to memory-critical registers as executable, or even inject attacker-forged code without writing to memory-critical registers.

To that end, we introduce *page table latching* for SMs. A latch is a protection mechanism that software can enable but not disable [34], [39], [40]. Consequently, we refer to page table latching as fixing the page tables so that software cannot change them until the next device reset. We consider a page table as *latched* if there is no virtual memory address that maps to the physical location of the page table and allows write access. In other words, the physical memory storing the page tables itself can only be read from but not written to [41].

If all page tables referenced by memory-critical registers are latched in this manner, the memory configuration cannot be changed any longer. This would either require modifying a memory-critical system register to, for example, exchange the set of page tables to a location where the attacker has write access to, or altering the existing page tables. As described in Section IV-B, no instructions that access memory-critical registers are executable in SM-, and the page tables are latched. If SM+ sets up the page tables to exclusively map its own code and data, these measures prevent SM- from accessing memory belonging to the TZOS or the TAs.

V. Context Protection

We motivate the necessity for protecting the CPU context of lower-privileged ELs. SM- saves and restores contexts when switching from the NW to the TZOS and vice versa. Since the contexts are saved to the data segment of SM-, an attacker that compromised SM- can tamper with the contexts to launch code-reuse attacks against the TZOS.

a) Saved CPU Context: To evaluate the attacker's capabilities, we first analyze the contents of a saved CPU context from a lower-privileged EL, as implemented by TF-A [19]. First, the CPU context includes the general purpose registers (*x0-x29*), as well as the link register (*LR*). The latter holds the address to jump to after the currently executing function returns. Additionally, EL1 system configuration registers of both worlds are part of the saved context. This is necessary since the NW OS and the TZOS configure them individually. Among others, these include the address to return to when switching to a userspace application (*ELR_EL1*), the saved stack pointers (*SP_EL1* and *SP_EL0*), the system configuration register to enable the MMU for EL1 (*SCTLR_EL1*), the page table base registers (*TTBR0_EL1* and *TTBR1_EL1*), and the address of the interrupt vector table for (S-)EL1 (*VBAR_EL1*).

Moreover, the context also includes some registers *exclusively* accessible to EL3. SM- saves these registers upon entering EL3- and restores them before leaving to a lower-privileged EL. Among others, these include the system configuration register that contains a bit to indicate the world (NW or SW) to return to after leaving EL3- (*SCR_EL3*), the exception link register that stores the address in a lower EL to return to after leaving EL3- (*ELR_EL3*), and the saved program status register that contains the CPU flags (e.g., zero flag and overflow flag) which are set when leaving EL3- (*SPSR_EL3*).

b) SMC-oriented Programming: Based on the registers that an SM- attacker can manipulate within the saved CPU context, we demonstrate a strategy for chaining existing code gadgets, as usual for code-reuse attacks [27]. We assume that the NW executes an SMC to switch to SM-, which saves the CPU context to SM- memory. Consequently, the saved *ELR_EL3* register contains the NW address of the SMC instruction that led to the context switch to SM-. An SM- attacker can now alter the saved value of *SCR_EL3* to trick the system into switching to the SW when SM- returns. Moreover, the attacker can alter the saved *ELR_EL3* register to divert the control flow to an arbitrary location in the TZOS. This allows an SM- attacker to execute *one* gadget. Analogous to code-reuse attacks, we refer to a gadget as a sequence of branchless instructions available in the TZOS, followed by an indirect branch instruction (i.e., an indirect jump, an indirect call, or a return instruction). However, an EL3 attacker cannot access the memory of the TZOS directly. Thus, the attacker cannot prepare a jump table, a call table, or a ROP slide in a writable memory region in the TZOS.

Therefore, we rely on a different chaining strategy. An SM- atacker can also change the link register *LR* in the saved context to the address of an SMC instruction. When returning to

the TZOS, the gadget executes and returns. The return jumps to the address in *LR*, which points to an SMC, yielding the control back to the SM. The attacker can access the results of the gadget from the saved general-purpose registers *x0-x29* and even re-arrange the general-purpose registers before calling the next gadget. By chaining gadgets, an attacker can execute complex instruction flows, depending on the available gadgets. We refer to this technique as SMC-oriented programming.

A. Self-Managed TZOS Context

We observe that the root cause of context-based attacks is the TZOS trusting the context restored by SM-. For this reason, we propose adjustments to the TZOS so that it saves its own general-purpose and system registers whenever switching to SM- and restores them when SM- switches to the TZOS.

We store the registers on the stack of the TZOS. The stack pointer, in turn, is stored at and restored from a fixed, core-local address in the TZOS, which is inaccessible to SM-. This ensures that neither the stack pointer nor the context it references can be tampered with by SM-. Saving and restoring every general-purpose register would prevent passing arguments between the SM and the TZOS. Therefore, we follow the ARM SMC64 calling convention [42] and use *x0–x6* for parameters.

We reserve memory in the TZOS to store the saved context and identify context switches from the TZOS to the SM in the source code of the TZOS. Before every SMC, we save the TZOS context to the reserved memory and clear the registers from potentially confidential values.

Typically, a TZOS exposes entry points, which are called by the SM when a TZOS service is requested (e.g., a TA should be executed). We modify the TZOS and prepend instructions to restore the TZOS context to these entry points. Given that SM- uses the legitimate entry points to the TZOS, the TZOS saves and restores its own general-purpose and system registers. Therefore, we describe a methodology to force SM- to use these entry points.

B. Return-Critical Registers

The saved CPU context in SM- also includes *ELR_EL3*, *SCR EL3*, and *SPSR_EL3*, registers that are only accessible at EL3. For these registers, we cannot transfer the responsibility of managing them to the TZOS, yet they can contribute to context-based attacks.

Definition. *A system register is considered return-critical if the hardware relies on it when switching from EL3 to S-EL1, prior to the execution of the first instruction in the TZOS.*

We identify *SCR_EL3*, *SPSR_EL3*, and *ELR_EL3* as return-critical. These registers are read by the hardware upon the execution of an Exception Return (ERET) instruction [35, p. C6-1920], which returns from a higher-privileged EL to a less-privileged one. Although accessible at EL1, we also classify *SCTLR_EL1* as return-critical. This is because the register controls the activation of the MMU at EL1, thereby influencing how the return address stored in *ELR_EL3* is interpreted by the hardware. Specifically, if the MMU for EL1 is enabled, the return address is treated as a virtual address, whereas *ELR_EL3* stores a physical address otherwise.

We propose *guards* to verify the return-critical registers thereby controlling the exact conditions under which SM- can switch to the TZOS. For example, we use the guards to enforce that SM- can only jump to the designated TZOS entry points. This requires certain values in *SCTLR_EL1* (i.e., deactivated MMU) and *ELR_EL3* (i.e., address of entry point) when the ERET instruction is executed. Entry guards (Section V-C) are executed at EL3-, directly *after* a transition from a lower EL to EL3-. Exit guards (Section V-D) are executed at EL3-, directly *before* a transition from EL3- to the TZOS.

C. Entry Guards

We design entry guards to first save the values of return-critical registers to the CPU context and then invalidate these registers. We refer to *invalidating* as writing values to the return-critical registers that prevent switching from SM- to the TZOS or any of its TAs, without updating the return-critical registers first (Section V-D).

We adjust the exception handlers in SM- and prepend the entry guard to each. Page table latching (Section IV) ensures that SM- code cannot be written, which prevents modifications to the entry guard. The address of the EL3 exception vectors depends on *VBAR_EL3*. We initialize the register in SM+ and verify that no write instruction to it is executable in SM-.

a) Masking Interrupts: We require the entry guards to execute without interruption. The ARMv8-A architecture guarantees that Interrupt Requests (IRQs) are masked upon taking an exception to EL3 and that they remain masked until they are explicitly unmasked [35, p. D1-5977]. We observe that SM firmware is non-preemptive. TF-A only enables external aborts, which result in a controlled system freeze. We adjust TF-A to also keep external aborts masked and verify that no write instructions to the registers (*DAIF* and *DAIFClr*), which hold the interrupt mask, are executable in SM-. Our modifications defer the external abort until we leave SM-, at which point the abort is immediately handled. In sum, these measures ensure that the entry guard is executed for each entry into SM-. We discuss security in Section VI.

b) Gadget Prevention: Critically, the entry guard is required to be free of gadgets that empower the attacker to write arbitrary values into return-critical registers. Otherwise, the very code that should prevent context-based attacks could be used to conduct them. Therefore, we carefully engineer a gadget-free sequence of instructions that cannot be used as a gadget to write malicious values into return-critical registers. Moreover, the sequence of instructions does not contain dynamic branching instructions.

Listing 1 shows a simplified version of the entry guard that is executed directly after switching from a lower-privileged EL to SM-, at the example of the return-critical register *SCR_EL3*. One important observation is that the values in the return-critical registers do not take effect immediately. Instead, they

only become active after exiting EL3 and returning to a lower EL. We rely on these characteristics to build the entry guards.

In lines 2–4, we first save *SCR_EL3* to the context. We only read the register with the `mrs` instruction, which cannot be used as a gadget to write the register.

We then invalidate the return-critical register. First, we load a safe value to a general-purpose register (line 7). A value of $0x1$ (in the first bit) of *SCR_EL3* indicates to switch to the NW when leaving SM-. Subsequently, we copy the value to the return-critical register (line 8). Without countermeasures, however, line 8 can be used as a gadget to write an arbitrary value to *SCR_EL3*, when conducting code-reuse attacks on SM-. Therefore, we directly *read* the register again in line 9 and compare the previously written value with the expected value (line 10). If they differ, a code-reuse attack such as ROP or JOP on the entry guard is detected and we directly induce a system freeze (line 12). If the check succeeds, the SMC is handled. Typically, this involves saving the rest of the EL1 registers, followed by calling the SMC handler (line 14).

We handle the other return-critical registers analogously.

Listing 1
SIMPLIFIED ENTRY GUARD.

```
1  vector_entry sync_exception_aarch64
2    // save return-critical registers to context
3    mrs x16, scr_el3 // read scr_el3
4    str x16, [sp, #EL3STATE_OFFSET + CTX_SCR_EL3]
5
6    // invalidate return-critical registers
7    mov x17, #0x1
8    msr scr_el3, x17 // write scr_el3
9    mrs x17, scr_el3 // read scr_el3
10   cmp x17, #0x1
11   b.eq check_valid
12   b . // endless loop
13 check_valid:
14   handle_sync_exception
15 end_vector_entry sync_exception_aarch64
```

D. Exit Guards

After executing the entry guard, the return-critical registers are invalidated (Section V-B) until they are updated by the exit guards. We use the exit guards to control which values SM- can write to return-critical registers, thereby enforcing the conditions for switching from SM- to the TZOS.

We prepend the exit guard's instructions before *every* occurrence of an ERET instruction in SM-, without any intervening dynamic branch instructions. By latching the page tables (Section IV), we prevent code modifications to the exit guards. By masking IRQs in SM- (Section V-C), we prevent an attacker from interrupting the exit guard.

a) Register Dependencies: We observe that we need to restrict the values of return-critical registers depending on other return-critical registers. The safe values of *SCTLR_EL1*, *SPSR_EL3*, and *ELR_EL3* depend on the value of *SCR_EL3*. If the latter indicates a switch to the NW, arbitrary values for the other return-critical registers are allowed. If we switch to the TZOS, however, we need to ensure a well-defined entry to mitigate context-based attacks. For example, we need to ensure that *ELR_EL3* can only be set to a TZOS entry point.

Listing 2 presents a simplified version of the exit guard macro at the example of *SCR_EL3* and *ELR_EL3*. We replace each ERET instruction with the exit guard macro, which the linker inlines at the corresponding locations. We sequentially restore return-critical registers while we verify if SM- tries to switch to the TZOS. We first restore *SCR_EL3* from the context (lines 2–3). Subsequently, we verify if the value written to *SCR_EL3* can indicate a switch to the TZOS (lines 4–6). If we switch to the NW, we jump to the *exit* label (line 7) and just execute an ERET (line 19).

If *SCR_EL3* indicates a switch to the SW, we also restore *ELR_EL3* (lines 10–11) and verify if the value that was written to the register matches the TZOS entry point (lines 12–14). To be more precise, the TZOS communicates its entry points to the SM during its initialization (t_3, Figure 2). SM+ saves the entry points to a dedicated segment that is read-only after the privilege drop in SM-. We compare the value of *ELR_EL3* with the value in that segment. Only if they match, we execute an ERET. We freeze the system if a discrepancy is detected between the value restored from the context and the value written to *ELR_EL3* (line 17).

Listing 2
SIMPLIFIED EXIT GUARD.

```
1  .macro exit_guard
2    ldr x18, [sp, #EL3STATE_OFFSET + CTX_SCR_EL3]
3    msr scr_el3, x18 // write scr_el3
4    mrs x16, scr_el3 // read scr_el3
5    and x16, x16, #0x1
6    cmp x16, #0x1 // check first bit
7    b.eq exit // return to NW
8
9    // return to TZOS
10   ldr x18, [sp, #EL3STATE_OFFSET + CTX_ELR_EL3]
11   msr elr_el3, x18 // write elr_el3
12   mrs x16, elr_el3 // read elr_el3
13   ldr x17, =0xA020F000 // TZOS entry point address
14   cmp x16, x17
15   b.eq exit
16 panic:
17   b . // endless loop
18 exit:
19   eret
20 .endm
```

We handle the other return-critical registers similarly. For *SCTLR_EL1*, we ensure that the MMU for S-EL1 is deactivated when transitioning to the TZOS. This guarantees an unambiguous interpretation of *ELR_EL3* as physical address during the switch. Upon entry, the TZOS restores its own registers and re-enables the MMU for S-EL1. For *SPSR_EL3*, which determines the target EL, we ensure that direct context switches from SM- to S-EL0 are prevented.

b) Gadget Interleaving: It is crucial that the exit guard does not contain gadgets that can be used to mount context-based attacks against the TZOS. Therefore, we carefully interleave the write gadgets such that an attacker can *either* switch from SM- to the NW *or* properly execute the exit guard and switch to the TZOS under controlled conditions.

We analyze the potential of code-reuse attacks such as ROP and JOP on the exit guard. The attacker cannot use the *ELR_EL3* gadget in line 11 to write an attacker-controlled

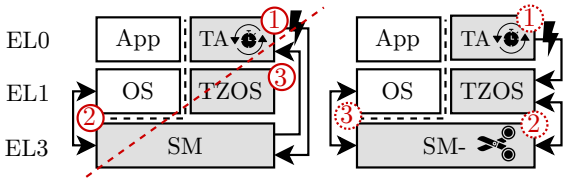

Fig. 3. IRQ routing models when a running TA is interrupted. On the left-hand side, the IRQ is taken to EL3, where the TA's context is saved. If the IRQ originated in the NW, it is forwarded to and handled by the NW OS. Afterward, the SM restores the TA's context and resumes execution in S-EL0. This routing model exposes TA contexts to the SM. Therefore, SH3ARS blocks this routing model. On the right-hand side, the TZOS receives the IRQ if a running TA is interrupted. The TZOS stores the TA's context and executes an SMC to switch to the SM, which forwards the IRQ. Afterward, the SM returns to the TZOS. SH3ARS enforces this routing model. Arrows indicate control flow. Flashes resemble interrupts.

value to *ELR_EL3*, as misuse of the gadget is detected (lines 12–15). Alternatively, the attacker can try to use the *SCR_EL3* gadget (line 3). However, *if* the attacker tries to program *SCR_EL3* so that a switch to the TZOS is triggered, *ELR_EL3* is restored and verified next, enforcing a legitimate value for *ELR_EL3*. The exit guard contains no dynamic branching instructions that could allow an attacker to bypass the handling of *ELR_EL3* after writing a value to *SCR_EL3*.

Finally, an attacker can directly jump to the ERET instruction (line 19). However, the entry guards (Section V-C) ensure that the return-critical registers are invalidated when switching to SM-. We ensure that, within SM-, there are no write gadgets for return-critical registers except for those in the guards.

E. TZOS Interrupt Routing

While SM- is non-preemptible, NW IRQs are generally enabled when the TZOS or a TA runs. If an IRQ occurs (e.g., a timer from the NW OS), the TZOS transfers control back to the NW via SM-. Masking IRQs in the TZOS is not a viable option, since the TAs would defer NW tasks for a long time.

IRQs can be exploited to leak information from confidential contexts [43]–[45]. Thus, SH3ARS requires a suitable routing model to process TZOS interrupts. We first illustrate a potential attack vector (Figure 3, left-hand side). Consider the case that a TA is running and an IRQ occurs. If the IRQ is directly routed to SM-, SM- saves the TA's general-purpose registers in the context, exposing the confidential TA contents to SM-, which can be compromised under our threat model (Section III). Moreover, an SM- attacker can also tamper with the saved context, mounting context-based attacks on the TAs.

We observe that the reason for this attack vector is that the TZOS has no chance to react to the IRQ. Therefore, we require SM- to configure the system so that IRQs are routed to a handler in the TZOS when they occur during the execution of the TZOS or a TA (Figure 3, right-hand side). Upon receiving an IRQ, the TZOS stores the TA's context in its own memory, which is inaccessible to SM- (Section IV). Subsequently, the TZOS can sanitize the register values before forwarding the request to SM- for further handling.

TABLE I
SH3ARS' TRUSTED COMPUTING BASE.

Entity	LoC [N]	Relative [%]
Stock Trusted Firmware-A (TF-A)	12 486	17.71
Stock OP-TEE TZOS	57 622	81.75
SH3ARS		
Memory Protection (Section IV)	74	0.10
Context Protection (Section V)		
Entry Guard	49	0.07
Exit Guard	77	0.11
Self-Managed Context (Section V-A)	180	0.26

Fortunately, this is one of the routing models supported by OP-TEE [32], the TZOS implementation we use. Nevertheless, it is essential to ensure that SM- is incapable of modifying the routing model.

Definition. *We define a system register as exception-critical if it affects the target EL of an exception that occurs during the execution of software at S-EL0 and S-EL1.*

We identify *SCR_EL3* as exception-critical. In particular, we identify the fields *IRQ*, *FIQ*, *EA*, and *RW*. These registers define whether interrupts, fast interrupts (high-priority interrupts), and external abort exceptions are routed to EL3 [35, p. D1-6002]. Notably, we already identified *SCR_EL3* as return-critical (Section V-B). Therefore, we extend the existing checks in the exit guards to ensure that exceptions arrive at the TZOS when either the TZOS or a TA is interrupted.

VI. SECURITY DISCUSSION

We evaluate the security of SH3ARS by discussing attacks and their mitigations. Moreover, we measure the TCB.

A. Trusted Computing Base

We require the source code of the SM and the TZOS to be available and implement SH3ARS by modifying and extending the source code of TF-A and OP-TEE, while using binary analysis to ensure certain properties on the generated binaries.

We use `cloc` (v1.82) to count the Lines of Code (LoC) of the source files that contributed to the machine code when compiled for our target platform [22] in the TF-A and the OP-TEE TZOS binary, as well as for our additions to the TCB. The results are shown in Table I. We do not consider the device's bootloader, as it remains unmodified. Overall, the components of SH3ARS introduce only minimal additions to the TCB. In total, SH3ARS adds 380 LoC (0.53%).

B. Attack Vectors

We derive Security Prerequisites (SPs) to systematically deal with attack vectors on SH3ARS and their mitigations.

SP1 SM+ must securely set up the system.
SP2 SM- must not map TZOS or TA memory.
SP3 The page tables of SM- must be latched at EL3-.
SP4 Memory-critical registers must be read-only at EL3-.

SP5 The TZOS must not rely on SM- to restore general-purpose registers and S-EL1 configuration registers.
SP6 Entry guards must be executed when switching to SM-.
SP7 Exit guards must be executed when switching to the TZOS.
SP8 If the TZOS or a TA is interrupted, the IRQ must be routed to the TZOS handler first.

a) Boot Time Attacks: SM+ is initialized early on in the system's boot process (Figure 2), directly after the bootloader. At this stage, the NW has not been initialized. By the time potentially malicious input from the NW is processed, the system has already transitioned to SM-. Moreover, secure boot (Section III) ensures the integrity of the bootloader and the SM binary when they are loaded to memory. Consequently, we argue that an adversary cannot feed malicious inputs to the bootloader or SM+ and the system is set up securely (SP1), while only SM- is exposed to an attacker.

b) Attacks on Memory Protection: An attacker that compromised SM- might attempt direct access to TZOS or TA memory. However, SM+ configures the page tables to map only SM- code and data and activates the MMU (SP2).

Alternatively, the attacker could attempt to alter the memory mappings, either by modifying the page tables or by changing a memory-critical register. However, the page tables are latched (SP3). We verify the correctness of the page tables and their proper latching by printing the tables as well as with practical tests. Specifically, we practically confirm that access faults occur when SM- attempts to (1) access unmapped memory, (2) write to read-only memory like the page tables, or (3) execute non-executable memory. These results give us strong confidence in the correct setup of the page tables.

Moreover, the attacker can attempt to modify a memory-critical register (e.g., to deactivate the MMU) by injecting code into SM-. However, writable regions in SM- cannot be executed, and executable regions are not writable. Because the page tables are latched, the attacker cannot alter the permissions. As a result, injecting code into SM- is not possible. Consequently, an attacker would have to utilize existing code to modify a memory-critical register (code-reuse attacks). On ARMv8.0-A, configuration registers can only be written using the `MSR` instruction [35, p. C5-808]. Through static analysis, we ensure that SM- does not contain any `MSR` instruction that writes to a memory-critical register. Moreover, we make sure that we did not miss a memory-critical register for ARMv8.0-A. The ARM reference manual provides a detailed list of the registers that are used for MMU configuration [35, p. D8-6440], which we use to derive memory-critical registers. Following related work [36]–[38], we are not aware of instructions that directly operate on physical addresses.

Additionally, an attacker cannot jump to an arbitrary byte within an instruction to forge new instructions (e.g., `MRS`), as it is possible on CISC architectures. This is because ARMv8.0-A enforces strict instruction alignment, with each instruction being exactly four bytes wide. This property has been previously leveraged for security [36]–[38]. Consequently, we can precisely verify that SM- contains no instruction that writes to a memory-critical register, and cannot be tricked into constructing one (SP4).

c) Attacks on Context Restoration: To mitigate context-based attacks like SMC-oriented programming (Section V), we modify the TZOS to save and restore its own general-purpose and the S-EL1 system registers. We validate our modifications by printing the registers before and after switching to the TZOS. Additionally, we temporarily alter SM- to overwrite the saved TZOS context, including the stack pointer, with a byte pattern and still successfully switch to the TZOS. This confirms that the TZOS does not depend on SM- to restore the context (SP5).

d) Attacks on Entry Guard: We propose entry guards (Section V-C) to invalidate return-critical registers, which could be used to foster context-based attacks otherwise. We require the entry guards to be executed unconditionally when switching to SM-, even if SM- is compromised. Therefore, we must guarantee that:

1) The attacker cannot modify the entry guard.
2) The attacker cannot alter the exception handlers' location.
3) The attacker cannot interrupt the entry guard.

The latched page tables ensure that SM- code cannot be written, which prevents modifications to the entry guard (1). The system register *VBAR_EL3* holds the vector base address for any exception that is taken to EL3 [35, p. D1-5976, D24-8491]. We initialize *VBAR_EL3* in SM+ and ensure that no write instructions to it are executable in SM-. Thus, the exception handlers are fixed (2), which also prevents attackers from redirecting execution to a different, but still valid, entry guard. The hardware guarantees to mask interrupts when switching to EL3 [35, p. D1-5977]. Moreover, we ensure that no writes to the interrupt mask registers (*DAIF* and *DAIFClr*) occur in SM-. There is no other register that overrules the *DAIF* register [36].

Synchronous exceptions (e.g., divide by zero) cannot be masked. We observe that no synchronous exceptions occur while TF-A executes under normal conditions. If they do, they indicate a system error. To counter an attacker from triggering a synchronous exception to interrupt the entry guard, we ensure that the handler for synchronous exceptions triggers a controlled system crash without any branching instructions.

Additionally, the entry guard contains no dynamic branches that could allow an attacker to exit before its completion by mounting code-reuse attacks. Therefore, the attacker cannot interrupt the entry guard (3). In sum, the entry guard is executed unconditionally (SP6).

e) Attacks on Exit Guards: Exit guards (Section V-D) control the return-critical registers when switching from SM- to the TZOS. We require exit guards to be executed whenever switching from SM- to the TZOS, even if SM- is compromised. We place the exit guard immediately before every ERET instruction in SM-. Using static analysis, we ensure that no executable ERET instruction exists in SM- without a preceding exit guard. As with the entry guards, memory protection prevents tampering with the exit guard's code,

and interrupt masking and careful engineering ensures that it cannot be interrupted during execution. Moreover, ERET is the only instruction that switches from EL3 to a less-privileged EL [46].

We carefully design the entry guards and exit guards to be free of gadgets that could be exploited in code-reuse attacks on SM-. Via gadget interleaving (Section V-D), the exit guards ensure that either SM- switches to the NW or SM- switches to the TZOS and undergoes verification of a proper TZOS entry. Through static analysis, we verify that no instructions capable of writing to return-critical registers are executable in SM- outside of those within the guards.

Code-reuse attacks on SM- in between the entry and exit guards are possible but ineffective, as all exit guards enforce a secure entry into the TZOS and preserve isolation. The TZOS can safely handle unexpected but valid entries.

Finally, we have strong indications that we did not miss a return-critical register. First, the ERET instruction does not rely on registers except those that we identified as return-critical [35, p. C6-1920]. Second, any further return-critical EL3 register would need to be saved and restored by TF-A. We find that only the EL3 registers *PMCR_EL3* and *ESR_EL3* are saved and restored in addition to the return-critical registers. *PMCR_EL3* controls access to performance counters in lower ELs, and the latter stores the reason (e.g., an SMC) for switching from a lower EL to SM-. Neither of them affects the context switch between SM- and the TZOS. In sum, the exit guard is executed before switching to the TZOS, even if SM- is compromised (SP7).

f) Interrupt-Based Attacks: SM- must not directly receive IRQs when the TZOS or a TA is interrupted. The exit guards ensure that `SCR_EL3` is configured correctly (Section V-E). With the correct configuration in `SCR_EL3`, the hardware guarantees to route exceptions directly to the TZOS when the TZOS or a TA is interrupted [35, p. D1-6002]. Since the exit guard runs before switching to the TZOS, a secure interrupt routing model is enforced (SP8).

g) Cache-Based Attacks: On ARMv8.0-A hardware, the SM and the TZOS share data and instruction caches, as well as the Translation Lookaside Buffer (TLB). This might enable SM- to mount cache-based attacks on the TZOS and its TAs. First, a compromised SM may attempt to poison the cache prior to switching to the TZOS, aiming at influencing the TZOS by causing it to read the injected instructions or data from the cache. Second, a compromised SM may attempt to directly read sensitive data of the TZOS or the TAs from the shared cache, thereby leaking data to the SM.

However, we note that latching the page tables for SM- and a TZOS working on its own address range can mitigate these attacks. Cache lines in the data and instruction cache are tagged by either their virtual address or their physical address. TLB cache lines are tagged by their virtual address. However, since the page tables of SM- are latched, SM- operates within a fixed virtual address range with predetermined mappings to physical addresses. Consequently, the set of cache tags that the SM can directly access is statically known.

The TZOS operates on a virtual and physical address range that is disjoint from the SM. Therefore, the cache tags that the TZOS and the TAs access do not overlap with those of SM-, which prevents direct cache-based attacks. Nonetheless, as both domains share the same physical cache, cache contention remains possible, potentially enabling cache-based side-channel attacks. We note that cache-based side channels represent a limitation of the ARM TrustZone TEE architecture [47], [48]. These vulnerabilities are not specific to SH3ARS and are out of scope.

VII. Performance Evaluation

We evaluate our proof-of-concept implementation of SH3ARS in terms of boot delay for the SM, as well as the runtime impact. We implement SH3ARS on an i.MX 8MQuad Evaluation Kit, which features an NXP i.MX8M Cortex-A53 CPU and a Cortex-M4 coprocessor. We aim for a realistic environment and base our implementation on the vendor-provided fork of TF-A, referred to as *imx-tfa*, which is based on TF-A v2.2. As a TZOS, we rely on OP-TEE 3.7 [32], a widely-used open-source TZOS [49]–[51]. In the NW, we run Linux 5.10. Our baseline is an unmodified system.

A. SM Boot Delay

We evaluate the SM boot time by measuring the interval between the starting TF-A (t_1, Figure 2) and the switch from SM to the NW (t_5, Figure 2) using the system counter register *CNTPCT_EL0*. We conduct 100 boot cycles.

TF-A takes 1.71s on average to boot with our baseline. With SH3ARS, we average 1.70 seconds, which we attribute to slight measurement inaccuracies. The standard deviation is insignificant. Thus, we conclude that SH3ARS does not induce notable boot time delay for TF-A. We deem this reasonable given that we do not notably increase the boot complexity of TF-A, but instead redistribute the existing logic across SM+ and SM-.

B. Runtime Performance Impact

a) World Switch Latency: We utilize the system counter *CNTPCT_EL0* to measure the latency of world switches in microbenchmarks. First, we measure the time elapsed between the execution of the last instruction in the NW OS prior to issuing an SMC and the first instruction executed in the SMC handler in SM-. This experiment is intended to quantify the overhead introduced by the entry guard (Section V-C). Second, we measure the time between issuing an ERET in SM- and executing the first instruction in the corresponding handler in the TZOS. This experiment is designed to quantify the overhead introduced by the exit guard (Section V-D).

For each experiment, we conduct $20,000$ runs. We measure a latency of 0.45 ± 0.06 μs for our baseline when entering SM-, compared to 3.30 ± 0.17 μs for a setup with our entry guards. Likewise, we determine a latency of 0.19 ± 0.06 μs for the baseline when entering SM-, and 7.77 ± 0.10 μs for a setup with our exit guards.

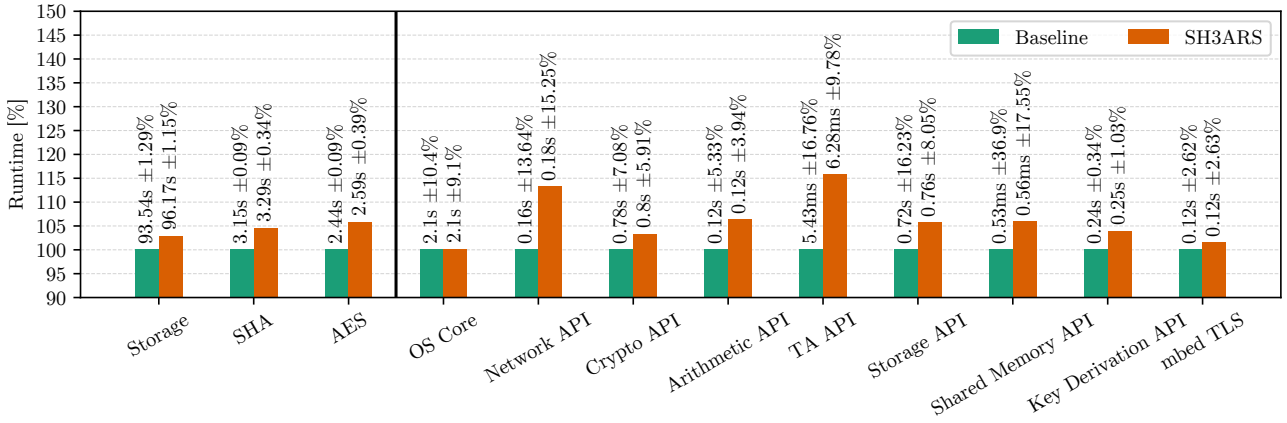

Fig. 4. Macrobenchmark results from the `xtest` benchmarking and regression test suite. The first three groups on the left show the results from the benchmarks, which primarily stress either the storage subsystem or the CPU (specifically, the *SHA* and *AES* tests). The groups on the right show results from regression tests per category. Bars represent percentual runtime relative to the baseline system, while labels indicate the absolute runtime along with the coefficient of variation, computed over 50 runs.

We observe that even a small number of instructions during the context switch can significantly increase the latency when measured in a microbenchmark. Given that more instructions are introduced during the exit guard, a proportionally greater increase is to be expected. This aligns with related research [36], which also observes high microbenchmark overheads when adding instructions to world switches. However, the results from microbenchmarks do not necessarily reflect the impact on real workloads. Therefore, we also conduct macrobenchmarks to assess SH3ARS' practical implications.

b) Benchmark Suite: Following the methodology of prior work [31], we utilize *xtest*, a regression testing and benchmarking suite for OP-TEE, comprising 95 tests across nine categories (e.g., secure storage, cryptography, and shared memory) as well as seven benchmarks targeting both CPU-bound and I/O-intensive TAs. For each experiment, we run the test 50 times, average the results, and calculate the overhead in regard to our baseline system as well as the coefficient of variation. The results are presented in Figure 4.

Overall, SH3ARS induces an average runtime performance overhead of 5.73% in the stress tests and the regression tests. We observe two outliers, namely the *Network API* (13.25%) and the *TA API* (15.71%). The latter shows very short execution times within the TA. Our microbenchmarks already indicate that SH3ARS introduces overhead for context switches. Thus, when the TA executes barely any instructions and directly returns, the context switch dominates, increasing the relative overhead.

We identify a similar cause for the *Network API* tests, which spawn a server in the NW and relay data between it and a TA using multiple threads, leading to frequent context switches. Three of the four tests use 64-byte buffers and show higher overhead on average (10.97%), while the test with a 16,384-byte buffer incurs only a 5.0% penalty.

In summary, we conclude that SH3ARS comes with an overhead of less than 6% for most workloads in practice.

VIII. TZOS Privilege Reduction

We described SH3ARS, a system architecture to deprivilege the SM on ARMv8.0-A-based devices, securing the confidential data of the TZOS and the TAs even when the SM is compromised. However, in default ARMv8.0-A-based systems, the TZOS also has unrestricted access to SW memory. Exploiting a vulnerability in the TZOS, an attacker can try to take over the NW OS (e.g., to backdoor the NW kernel [29], [52]), control other TAs (e.g., to extract secrets [53], [54]), or hijack the SM (e.g., to hide malware [55]).

While SH3ARS targets deprivileging the SM, this section describes the conceptional integration of ReZone [31], a state-of-the-art approach to reducing the TZOS privileges. Together, they ensure *mutual* isolation between the SM and the TZOS. To the best of our knowledge, this has not been achieved to date. We provide an overview of ReZone and the conceptional modifications required for integration with SH3ARS. For a detailed explanation of ReZone, we refer to the original paper.

A. Overview of ReZone

ReZone confines the TZOS in a sandboxed domain, referred to as a "zone", with access only to its private resources (Figure 5). To achieve this level of isolation, ReZone leverages hardware features beyond the ARMv8.0-A standard, which are available on most, but not all, devices.

a) Auxiliary Control Unit (ACU): Many ARMv8.0-based systems implement co-processors to offload specific tasks from the main CPU to improve efficiency, performance, or power consumption. ReZone requires such a co-processor and refers to it as an ACU. The i.MX 8MQuad Evaluation Kit, which we use in our implementation, includes a Cortex-M4 co-processor that can serve as the ACU.

b) Platform Partition Controller (PPC): Moreover, ReZone requires a PPC, which is a programmable protection controller that can restrict access to ranges of physical memory

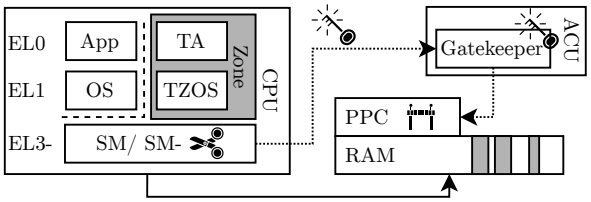

Fig. 5. ReZone partitions S-EL1 and S-EL0 into "zones" (grey background). ReZone requires a PPC, which, alongside TrustZone-based partitioning, can also divide the physical memory into partitions. The PPC prevents the zone from accessing other memory. Only a system co-processor, the ACU, can reconfigure the PPC. To switch between zones, the SM sends a message to the gatekeeper running on the ACU. The SM passes a token (key) as an authentication feature, preventing other ELs from initiating zone switches themselves. Solid arrows indicate memory access. Dotted arrows indicate control flow between system components.

or peripherals. This protection is orthogonal to the TrustZone-based partitioning. While TrustZone allocates memory and peripherals as SW or NW resources, depending on the CPU state, the PPC manages resources between components in multi-processor systems (e.g., the CPU and co-processors).

c) Zone Switching: ReZone requires a hardware configuration where only the ACU can configure the PPC. ReZone employs the ACU to run software referred to as *gatekeeper*. Before switching to the TZOS, the SM sends a message to the gatekeeper, instructing it to restrict the main CPU's access to the memory and peripherals based on the PPC. When the TZOS returns, the SM sends another message to the gatekeeper to lift the restrictions again.

d) Secret Token: Typically, the main CPU and the ACU communicate via a message queue. An attacker in S-EL1 could attempt to craft a malicious message to the gatekeeper to disable PPC protection. To prevent such attacks, the SM and the gatekeeper share a secret token (illustrated as key, Figure 5), which is initialized at boot time. The SM stores this token in the system register *TPIDR_EL3*, which is intended to store thread-identifying information and is not used within TF-A. Critically, only software executing at EL3 can access this register. As a result, a compromised S-EL1 cannot retrieve the secret token or forge a message to the gatekeeper.

B. Adapting ReZone to SH3ARS

ReZone expects the SM to be free of software vulnerabilities. In contrast, we assume that an attacker manages to compromise the SM firmware. Therefore, we describe the necessary modifications to integrate ReZone with SH3ARS.

A secure integration must guarantee switching to the TZOS zone when entering the TZOS, even if the SM is compromised. To that end, we propose to embed the zone-switching into our entry and exit guard. The exit guard enables the TZOS zone when switching to the TZOS, while the entry guard disables it when switching back to SM-. Since the guards are executed unconditionally, zone switching – and, thus, TZOS privilege reduction – is guaranteed even if SM- is compromised.

Moreover, the secret token that is shared between the SM and the gatekeeper must be protected from an attacker. With SH3ARS, SM+ can initialize the secret token, and, like ReZone, store it in the otherwise unused *TPIDR_EL3* register. Analogous to memory-critical or return-critical registers, we can ensure that SM- does not include executable instructions that access *TPIDR_EL3*, with the exception of a *switch_zone* macro shown in Listing 3.

Listing 3
SIMPLIFIED MESSAGE PASSING TO THE GATEKEEPER.

```
1  .macro switch_zone
2    mrs x16, tpidr_el3 // load secret token
3    ldr x17, #0x30AA0000 // message queue address
4    str x16, [x17] // write token to message queue
5
6    wait_msg:
7    ldr x16, #0x30AA0020 // status address
8    ldr x16, [x16] // load queue status
9    tst x16, #0x8000000 // check status
10   b.eq wait_msg // wait until received
11
12   str xzr, [x17] // clear secret from queue
13 .endm
```

The macro, which we envision to be integrated into the guards, reads the secret token to a register (line 2), followed by sending the token to the gatekeeper (lines 3–4). SM- loops until the gatekeeper confirms the zone switch (lines 7–10). Finally, SM- clears the token from the message queue (line 12). An SM- attacker cannot leak the token, even when mounting code-reuse attacks. This is because the macro removes any relics, interrupts are masked, and the macro does not contain dynamic branches.

C. Complementing Techniques

While ReZone can be used to deprivilege the TZOS, it is not capable of deprivileging the SM. This is because there is no hardware layer with higher privileges than the SM to supervise and verify the switches between the zones. To mitigate this issue, SH3ARS employs temporal isolation between SM+ and SM-. Conversely, temporal isolation is not feasible for TZOS privilege reduction, as the TZOS typically requires dynamic memory reconfiguration at runtime. Consequently, the techniques of TZOS deprivileging and SM privilege reduction are complementary rather than competing. In combination, they enable mutual isolation of the SM and the TZOS, thereby containing vulnerabilities within their respective domains.

IX. Related Work

a) Kernel Privilege Reduction: Several works explore TEE-based or hypervisor-based protection mechanisms for kernels [37], [38], [56]–[58]. These prevent the kernel from modifying its memory mappings. Instead, the kernel must request changes from a higher-privileged component. From these works, we primarily adopt the idea of leveraging the aligned, fixed-size ARMv8 instructions and non-writable memory to verify the absence of instructions, such as write access to critical registers. In contrast to these, we deprivilege the SM, which is already the highest privilege layer on the system.

Other publications explore intra-level privilege separation mechanisms between a memory-supervising and a memory-constrained component, both executing with kernel-level privileges [36], [59], hypervisor-level privileges [60], or even with TZOS privileges [61], [62]. These works facilitate entry and exit gates, which inspired our guards. In contrast, we utilize the guards not for memory isolation, but to mitigate context-based attacks on the TZOS and TAs, even in the event of a compromise at the highest privilege level. SH3ARS achieves memory protection for the SM through page table latching, inspired by the concept of latching ranges in persistent storage [34], [39], [40], as well as by HyperSafe [41], which locks down hypervisor code. None of these works were designed to confine the otherwise omnipotent SM on ARMv8.0-A, as SH3ARS does.

b) Privilege Separation on RISC-V: RISC-V TEEs also introduce a software component operating at the highest privilege level, referred to as the *security monitor* [63]–[65]. Sharing a similar motivation, Kuhne et al. split the security monitor in two components, one containing firmware and another dedicated to memory protection [66]. They assume that the prior is free of vulnerabilities and describe the runtime interface between the components. We also split software at the highest privilege level into two components, namely SM+ and SM-. However, we rely on temporal isolation. SM+ sets up memory protection during boot and is never exposed to potentially malicious input, whereas SM- is exposed to such input but cannot lift the memory protection. Moreover, Kuhne et al. rely on a just recently ratified extension to the RISC-V hardware (ePMP). SH3ARS relies on standard ARMv8.0-A hardware, making it portable across existing devices.

c) User Space Enclaves: User space enclaves aim at protecting applications from an untrusted OS that manages them. Intel SGX [67], [68] enables dedicated hardware-enforced enclaves, while a stream of research focuses on repurposing memory protection techniques to enable user space enclaves [69]–[77]. In contrast, we aim to protect TAs running on top of a TZOS, as deployed on millions of real-world devices based on ARM TrustZone [29].

d) Confidential Computing Architectures: Research and industry have proposed a lift-and-shift approach to porting entire virtual machines into TEEs, isolating them from potentially untrusted hypervisors. By now, hardware support for confidential virtual machines is widely available, with techniques like AMD SEV-SNP [78], Intel TDX [79], ARMv9 with ARM CCA [25], and RISC-V with CoVE [80]. These approaches typically introduce new operating modes for isolating the memory management of the hypervisor from the untrusted rest and protecting the memory of the confidential VMs. Moreover, there is research on how to achieve similar guarantees without dedicated hardware support [81]–[83]. In contrast, we do not introduce new abstractions for TEE-based workloads but focus on securing existing ones by deprivileging SM software that could bypass TEE-based isolation.

e) Code-Reuse Attack Prevention and Detection: Other related work focuses on removing gadgets that can be used for code-reuse attacks or detecting such attacks, either in user-space software [84]–[86] or kernels [87]. However, completely preventing gadgets is challenging [88], [89]. Therefore, we adopt a fine-grained approach, identifying gadgets that could potentially bypass our memory and context protection, such as writes to critical registers. For these locations, we carefully engineer code to prevent misuse in code-reuse attacks.

X. Limitations and Future Work

a) Static SM Memory Layout: TF-A, at its core, does not require dynamic changes to its memory mappings after it is set up. However, a few of the runtime services do reconfigure the mappings during the runtime, even though this is technically not necessary. We analyze the source code of the TF-A versions from the past eight years (v2.0–v2.13) and identify a total of four such services (`opteed` [90], `trusty` [91], `qti` [92], and `widevine` [93]). In particular, `opteed` is a so-called dispatcher, which handles transitions into and out of OP-TEE. While OP-TEE supports late initialization requiring dynamic changes to the mappings, launching it during SM firmware setup (Section IV-A) only uses static mappings. Our evaluation confirms full functionality. Thus, no changes to `opteed` are required to make it functional with SH3ARS.

If required, the other three services need to be patched to be compatible with SH3ARS. However, we argue that this is practically feasible and involves mapping the memory already during the initialization phase of TF-A. Vendors aiming to adopt SH3ARS would need to make these adjustments.

While TF-A does not require dynamic changes to the memory mappings, other (proprietary) SM firmware may inherently rely on them. In these cases, same-level privilege separation [36], [66] could be applied. With such an approach, the memory-managing component is isolated from the rest of the SM firmware. However, the former still has full access to the memory, making it a prime target for attacks while being difficult to secure. Instead, we argue that latching the memory mappings of the most privileged software component, which should perform only minimal tasks nevertheless, is a valuable security trade-off.

b) Practical ReZone Integration: We describe how the central building blocks of ReZone [31] can be conceptually integrated into SH3ARS to achieve mutual isolation between the SM and the TZOS (Section VIII). However, an implementation of a combined proof-of-concept is left for future work.

c) Vulnerabilities in TZOS and TAs: We focus on attackers who escalate privileges from the NW to SM- by exploiting firmware vulnerabilities [1]. Another attack vector involves directly compromising the TZOS or a TA through its exposed interfaces or memory shared with the NW [94], [95]. While our focus is on SM privilege reduction, securing the TZOS and the TAs is and remains crucial. Existing works in this field complement ours [96]–[98].

d) Upcoming ARM Architectures: While we implement SH3ARS to protect the TZOS and TAs on ARMv8.0-based devices, ARMv8.4-A introduced virtualization for the SW, which provides new ways to prevent the TZOS from accessing SM

memory (as dealt with by ReZone). However, the SM remains omnipotent, with unrestricted access to memory. Therefore, SH3ARS remains applicable under the same conditions.

ARMv9 also features a SM at EL3. In contrast to ARMv8-A, where the SM runs in the SW, the SM of an ARMv9 device operates in a dedicated *root world*, separated from the SW. Nevertheless, the SM retains full control over memory. An adaption of SH3ARS to these architectures is future work.

XI. Conclusion

We present SH3ARS, a software architecture for ARMv8.0-A devices that addresses critical vulnerabilities in real-world SMs by reducing their privileges and reinforcing isolation for the ARM TrustZone TEE. To the best of our knowledge, SH3ARS is the first design to reduce the privileges for ARMv8.0-A SMs. SH3ARS ensures that the TZOS and TAs remain isolated, even if an attacker compromises the otherwise omnipotent SM firmware.

We adjust the SM firmware to irrevocably relinquish the privileges to access memory outside its designated address space. However, even without direct memory access, an SM-level attacker can try to exploit saved TZOS contexts to launch code-reuse attacks, dubbed SMC-oriented programming. Thus, we introduce guards, carefully crafted, gadget-free code sequences, that supervise the context switch to and from the TZOS. Consequently, SH3ARS maintains strong isolation for the TEE, even under an adverse attacker model.

We implement SH3ARS on real hardware using only standard ARMv8.0-A features, ensuring compatibility with a broad array of devices. SH3ARS induces a negligible overhead in terms of boot time and a runtime performance overhead of less than 6% for most workloads.

Acknowledgments

We thank the reviewers and our shepherd for their helpful feedback. This research was partly supported by the German Federal Ministry of Research, Technology, and Space (BMFTR) as part of the CELTIC-NEXT project SUSTAINET-inNOvAte ('Frictionless, secure, and resilient communication networks for the dynamic digital world", Förderkennzeichen "16KIS2258").

Author Contribution Statement

This section uses the CRediT taxonomy (https://credit.niso.org/). **Jonas Röckl:** Conceptualization, Methodology, Validation, Visualization, Software, Writing – Original Draft, Writing - Review & Editing, **Julian Funk:** Software, Validation, Writing - Review & Editing, **Matti Schulze:** Writing - Review & Editing, **Tilo Müller:** Supervision, Writing - Review & Editing

References

[1] C. Lindenmeier, M. Payer, and M. Busch, "EL3XIR: Fuzzing COTS Secure Monitors," in *33rd USENIX Security Symposium, USENIX SEC '24, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/lindenmeier

[2] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems," in *2020 IEEE Symposium on Security and Privacy, IEE S&P '20, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 1416–1432. [Online]. Available: https://doi.org/10.1109/SP40000.2020.00061

[3] S. Pinto, J. Pereira, T. Gomes, M. Ekpanyapong, and A. Tavares, "Towards a TrustZone-Assisted Hypervisor for Real-Time Embedded Systems," *IEEE Comput. Archit. Lett.*, vol. 16, no. 2, pp. 158–161, 2017. [Online]. Available: https://doi.org/10.1109/LCA.2016.2617308

[4] H. Park, S. Zhai, L. Lu, and F. X. Lin, "StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone," in *USENIX Annual Technical Conference, USENIX ATC '19, Renton, WA, USA, July 10-12, 2019*. USENIX Association, 2019, pp. 537–554. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/park-heejin

[5] Apple Inc. (2020, June) Apple announces Mac transition to Apple Silicon. [Online]. Available: https://www.apple.com/newsroom/2020/06/apple-announces-mac-transition-to-apple-silicon/

[6] J. Kalyanasundaram and Y. Simmhan, "ARM Wrestling with Big Data: A Study of Commodity ARM64 Server for Big Data Workloads," in *24th IEEE International Conference on High Performance Computing, HiPC '17, Jaipur, India, December 18-21, 2017*. IEEE Computer Society, 2017, pp. 203–212. [Online]. Available: https://doi.org/10.1109/HiPC.2017.00032

[7] F. Meneghello, M. Calore, D. Zucchetto, M. Polese, and A. Zanella, "IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8182–8201, 2019.

[8] S. Herwig, K. Harvey, G. Hughey, R. Roberts, and D. Levin, "Measurement and Analysis of Hajime, a Peer-to-peer IoT Botnet," in *26th Annual Network and Distributed System Security Symposium, NDSS '19, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/measurement-and-analysis-of-hajime-a-peer-to-peer-iot-botnet/

[9] M. Antonakakis, T. April, M. D. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, D. Kumar, C. Lever, Z. Ma, J. Mason, D. Menscher, C. Seaman, N. Sullivan, K. Thomas, and Y. Zhou, "Understanding the Mirai Botnet," in *26th USENIX Security Symposium, USENIX SEC '17, Vancouver, BC, Canada, August 16-18, 2017*. USENIX Association, 2017, pp. 1093–1110. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis

[10] M. D. Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-Capable IoT Malwares: Comparative Analysis and Mirai Investigation," *Secur. Commun. Networks*, vol. 2018, pp. 7 178 164:1–7 178 164:30, 2018. [Online]. Available: https://doi.org/10.1155/2018/7178164

[11] M. L. Vásquez, G. Bavota, and C. Escobar-Velasquez, "An empirical study on Android-related vulnerabilities," in *14th International Conference on Mining Software Repositories, MSR '17, Buenos Aires, Argentina, May 20-28, 2017*. IEEE Computer Society, 2017, pp. 2–13. [Online]. Available: https://doi.org/10.1109/MSR.2017.60

[12] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek, "Linux kernel vulnerabilities: state-of-the-art defenses and open problems," in *2nd Asia Pacific Workshop on Systems, APSys '11, Shanghai, China, July 11-12, 2011*. ACM, 2011, p. 5. [Online]. Available: https://doi.org/10.1145/2103799.2103805

[13] X. Zou, G. Li, W. Chen, H. Zhang, and Z. Qian, "SyzScope: Revealing High-Risk Security Impacts of Fuzzer-Exposed Bugs in Linux kernel," in *31st USENIX Security Symposium, USENIX SEC '22, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 2022, pp. 3201–3217. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/zou

[14] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 130:1–130:36, 2019. [Online]. Available: https://doi.org/10.1145/3291047

[15] Z. Ning and F. Zhang, "Understanding the Security of ARM Debugging Features," in *2019 IEEE Symposium on Security and Privacy, IEEE S&P '19, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 602–619. [Online]. Available: https://doi.org/10.1109/SP.2019.00061

[16] J. Röckl, A. Wagenhäuser, and T. Müller, "Veto: Prohibit Outdated Edge System Software from Booting," in *9th International Conference on Information Systems Security and Privacy, ICISSP '23, Lisbon, Portugal, February 22-24, 2023*. SciTePress, 2023, pp. 46–57. [Online]. Available: https://doi.org/10.5220/0011627700003405

[17] T. Groß, M. Busch, and T. Müller, "One key to rule them all: Recovering the master key from RAM to break Android's file-based encryption," *Digit. Investig.*, vol. 36 Supplement, p. 301113, 2021. [Online]. Available: https://doi.org/10.1016/j.fsidi.2021.301113

[18] X. Zheng, L. Yang, J. Ma, G. Shi, and D. Meng, "TrustPAY: Trusted mobile payment on security enhanced ARM TrustZone platforms," in *IEEE Symposium on Computers and Communication, ISCC '16, Messina, Italy, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 456–462. [Online]. Available: https://doi.org/10.1109/ISCC.2016.7543781

[19] ARM Limited. (2025) Trusted Firmware-A. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware

[20] ARM Limited. (2024) Power State Coordination Interface. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware

[21] B. Cook, K. Khazem, D. Kroening, S. Tasiran, M. Tautschnig, and M. R. Tuttle, "Model checking boot code from AWS data centers," *Formal Methods Syst. Des.*, vol. 57, no. 1, pp. 34–52, 2021. [Online]. Available: https://doi.org/10.1007/s10703-020-00344-2

[22] J. Röckl, N. Bernsdorf, and T. Müller, "TeeFilter: High-Assurance Network Filtering Engine for High-End IoT and Edge Devices based on TEEs," in *19th ACM Asia Conference on Computer and Communications Security, ASIA CCS '24, Singapore, July 1-5, 2024*. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3634737.3637643

[23] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. A. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: formal verification of an OS kernel." in *22nd ACM Symposium on Operating Systems Principles 2009, SOSP '09, Big Sky, Montana, USA, October 11-14, 2009*. ACM, 2009, pp. 207–220. [Online]. Available: https://doi.org/10.1145/1629575.1629596

[24] A. C. J. Fox, G. Stockwell, S. Xiong, H. Becker, D. P. Mulligan, G. Petri, and N. Chong, "A Verification Methodology for the Arm® Confidential Computing Architecture: From a Secure Specification to Safe Implementations," *Proc. ACM Program. Lang.*, vol. 7, no. OOPSLA1, pp. 376–405, 2023. [Online]. Available: https://doi.org/10.1145/3586040

[25] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and Verification of the Arm Confidential Compute Architecture," in *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI '22, Carlsbad, CA, USA, July 11-13, 2022*. USENIX Association, 2022, pp. 465–484. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/li

[26] W. Ozga, "Towards a Formally Verified Security Monitor for VM-based Confidential Computing," in *12th International Workshop on Hardware and Architectural Support for Security and Privacy, HASP '23, Toronto, Canada, 29 October 2023*. ACM, 2023, pp. 73–81. [Online]. Available: https://doi.org/10.1145/3623652.3623668

[27] R. Roemer, E. Buchanan, H. Shacham, and S. Savage, "Return-Oriented Programming: Systems, Languages, and Applications," *ACM Trans. Inf. Syst. Secur.*, vol. 15, no. 1, pp. 2:1–2:34, 2012. [Online]. Available: https://doi.org/10.1145/2133375.2133377

[28] T. K. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang, "Jump-oriented programming: a new class of code-reuse attack," in *6th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '11, Hong Kong, China, March 22-24, 2011*. ACM, 2011, pp. 30–40. [Online]. Available: https://doi.org/10.1145/1966913.1966919

[29] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel, and G. Vigna, "BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments," in *24th Annual Network and Distributed System Security Symposium, NDSS '17, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/boomerang-exploiting-semantic-gap-trusted-execution-environments/

[30] ARM Limited. (2024) The Most Widely Used Low-Power Processor. [Online]. Available: https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a53

[31] D. Cerdeira, J. Martins, N. Santos, and S. Pinto, "ReZone: Disarming TrustZone with TEE Privilege Reduction," in *31st USENIX Security Symposium, USENIX SEC '22, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 2022, pp. 2261–2279. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/cerdeira

[32] Linaro Limited. (2024) OP-TEE. [Online]. Available: https://github.com/OP-TEE/optee_os

[33] G. Beniamini. (2016) Unlocking the Motorola Bootloader. [Online]. Available: http://bits-please.blogspot.com/2016/02/unlocking-motorola-bootloader.html

[34] M. Xu, M. Huber, Z. Sun, P. England, M. Peinado, S. Lee, A. Marochko, D. Mattoon, R. Spiger, and S. Thom, "Dominance as a New Trusted Computing Primitive for the Internet of Things," in *2019 IEEE Symposium on Security and Privacy, IEEE S&P '19, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1415–1430. [Online]. Available: https://doi.org/10.1109/SP.2019.00084

[35] ARM Limited, *ARM Architecture Reference Manual ARMv8, for ARMv8-A Architecture Profile*, ARM Limited, 2022, ARM DDI 0487I.a. [Online]. Available: https://developer.arm.com/documentation/ddi0487/latest

[36] A. M. Azab, K. Swidowski, R. Bhutkar, J. Ma, W. Shen, R. Wang, and P. Ning, "SKEE: A lightweight Secure Kernel-level Execution Environment for ARM," in *23rd Annual Network and Distributed System Security Symposium, NDSS '16, San Diego, California, USA, February 21-24, 2016*. The Internet Society, 2016. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2017/09/skee-lightweight-secure-kernel-level-execution-environment-for-arm.pdf

[37] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen, "Hypervision Across Worlds: Real-time Kernel Protection from the ARM TrustZone Secure World," in *2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 2014, pp. 90–102. [Online]. Available: https://doi.org/10.1145/2660267.2660350

[38] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture," *CoRR*, vol. abs/1410.7747, 2014. [Online]. Available: http://arxiv.org/abs/1410.7747

[39] J. Röckl, M. Protsenko, M. Huber, T. Müller, and F. C. Freiling, "Advanced System Resiliency Based on Virtualization Techniques for IoT Devices," in *37th Annual Computer Security Applications Conference, ACSAC '21, Virtual Event, USA, December 6 - 10, 2021*. ACM, 2021, pp. 455–467. [Online]. Available: https://doi.org/10.1145/3485832.3485836

[40] M. Huber, S. Hristozov, S. Ott, V. Sarafov, and M. Peinado, "The Lazarus Effect: Healing Compromised Devices in the Internet of Small Things," in *15th ACM Asia Conference on Computer and Communications Security, ASIA CCS '20, Taipei, Taiwan, October 5-9, 2020*. ACM, 2020, pp. 6–19. [Online]. Available: https://doi.org/10.1145/3320269.3384723

[41] Z. Wang and X. Jiang, "HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity," in *2010 IEEE Symposium on Security and Privacy, IEEE S&P '10, 16-19 May 2010, Berleley/Oakland, California, USA*. IEEE Computer Society, 2010, pp. 380–395. [Online]. Available: https://doi.org/10.1109/SP.2010.30

[42] ARM Limited. (2013) SMC Calling Convention. [Online]. Available: https://developer.arm.com/documentation/den0028/a/

[43] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi, and S. Shinde, "HECKLER: Breaking Confidential VMs with Malicious Interrupts," in *33rd USENIX Security Symposium, USENIX SEC '24, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/schl%C3%BCter

[44] J. V. Bulck, F. Piessens, and R. Strackx, "SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control," in *2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*. ACM, 2017, pp. 4:1–4:6. [Online]. Available: https://doi.org/10.1145/3152701.3152706

[45] B. Schlüter, S. Sridhara, A. Bertschi, and S. Shinde, "WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP," in *2024 IEEE Symposium on Security and Privacy, IEEE S&P '24, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 4220–4238. [Online]. Available: https://doi.org/10.1109/SP54263.2024.00262

[46] Z. Ning and F. Zhang, "Ninja: Towards Transparent Tracing and Debugging on ARM," in *26th USENIX Security Symposium, USENIX SEC '17, Vancouver, BC, Canada, August 16-18, 2017*. USENIX Association, 2017, pp. 33–49. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ning

[47] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices," *IACR Cryptol. ePrint Arch.*, p. 980, 2016. [Online]. Available: http://eprint.iacr.org/2016/980

[48] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard, "ARMageddon: Cache Attacks on Mobile Devices," in *25th USENIX Security Symposium, USENIX SEC '16, Austin, TX, USA, August 10-12,*

2016. USENIX Association, 2016, pp. 549–564. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lipp

[49] F. Fleischer, M. Busch, and P. Kuhrt, "Memory corruption attacks within Android TEEs: a case study based on OP-TEE," in *15th International Conference on Availability, Reliability and Security, ARES '20, Virtual Event, Ireland, August 25-28, 2020*. ACM, 2020, pp. 53:1–53:9. [Online]. Available: https://doi.org/10.1145/3407023.3407072

[50] C. Göttel, P. Felber, and V. Schiavoni, "Developing Secure Services for IoT with OP-TEE: A First Look at Performance and Usability," in *Distributed Applications and Interoperable Systems, DAIS '19, Kongens Lyngby, Denmark, June 17-21, 2019*, ser. Lecture Notes in Computer Science, vol. 11534. Springer, 2019, pp. 170–178. [Online]. Available: https://doi.org/10.1007/978-3-030-22496-7_11

[51] H. Yang and M. Lee, "Demystifying ARM TrustZone TEE Client API using OP-TEE," in *9th International Conference on Smart Media and Applications, SMA '20, Jeju, Republic of Korea, September 17 - 19, 2020*. ACM, 2020, pp. 325–328. [Online]. Available: https://doi.org/10.1145/3426020.3426113

[52] G. Beniamini. (2016) War of the Worlds - Hijacking the Linux Kernel from QSEE. [Online]. Available: https://bits-please.blogspot.com/2016/05/war-of-worlds-hijacking-linux-kernel.html

[53] Gal Beniamini. (2016) Extracting Qualcomm's KeyMaster Keys - Breaking Android Full Disk Encryption. [Online]. Available: https://bits-please.blogspot.com/2016/06/extracting-qualcomms-keymaster-keys.html

[54] M. Busch, J. Westphal, and T. Müller, "Unearthing the TrustedCore: A Critical Review on Huawei's Trusted Execution Environment," in *14th USENIX Workshop on Offensive Technologies, WOOT '20, August 11, 2020*. USENIX Association, 2020. [Online]. Available: https://www.usenix.org/conference/woot20/presentation/busch

[55] M. Schulze, C. Lindenmeier, J. Röckl, and F. C. Freiling, "IlluminaTEE: Effective Man-At-The-End Attacks from within ARM TrustZone," in *2024 Workshop on Research on offensive and defensive techniques in the context of Man At The End (MATE) attacks, Salt Lake City, UT, USA, October 14-18, 2024*. ACM, 2024, pp. 11–21. [Online]. Available: https://doi.org/10.1145/3689934.3690838

[56] B. D. Payne, M. Carbone, M. I. Sharif, and W. Lee, "Lares: An Architecture for Secure Active Monitoring Using Virtualization," in *2008 IEEE Symposium on Security and Privacy, IEEE S&P '08, 18-21 May 2008, Oakland, California, USA*. IEEE Computer Society, 2008, pp. 233–247. [Online]. Available: https://doi.org/10.1109/SP.2008.24

[57] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," in *21st ACM Symposium on Operating Systems Principles 2007, SOSP '07, Stevenson, Washington, USA, October 14-17, 2007*. ACM, 2007, pp. 335–350. [Online]. Available: https://doi.org/10.1145/1294261.1294294

[58] D. Kwon, J. Seo, Y. Cho, B. Lee, and Y. Paek, "PrOS: Light-Weight Privatized Secure OSes in ARM TrustZone," *IEEE Trans. Mob. Comput.*, vol. 19, no. 6, pp. 1434–1447, 2020. [Online]. Available: https://doi.org/10.1109/TMC.2019.2910861

[59] N. Dautenhahn, T. Kasampalis, W. Dietz, J. Criswell, and V. S. Adve, "Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation," in *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '15, Istanbul, Turkey, March 14-18, 2015*. ACM, 2015, pp. 191–206. [Online]. Available: https://doi.org/10.1145/2694344.2694386

[60] L. Shi, Y. Wu, Y. Xia, N. Dautenhahn, H. Chen, B. Zang, and J. Li, "Deconstructing Xen," in *Annual Network and Distributed System Security Symposium, NDSS '17, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/deconstructing-xen/

[61] W. Li, Y. Xia, L. Lu, H. Chen, and B. Zang, "TEEv: virtualizing trusted execution environments on mobile platforms," in *15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE '19, Providence, RI, USA, April 14, 2019*. ACM, 2019, pp. 2–16. [Online]. Available: https://doi.org/10.1145/3313808.3313810

[62] Y. Cho, D. Kwon, H. Yi, and Y. Paek, "Dynamic Virtual Address Range Adjustment for Intra-Level Privilege Separation on ARM," in *Annual Network and Distributed System Security Symposium, NDSS '17, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/dynamic-virtual-address-range-adjustment-intra-level-privilege-separation-arm/

[63] A. Waterman, "Design of the RISC-V Instruction Set Architecture," Ph.D. dissertation, University of California, Berkeley, USA, 2016. [Online]. Available: https://www.escholarship.org/uc/item/7zj0b3m7

[64] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanovic, and D. Song, "Keystone: an open framework for architecting trusted execution environments," in *15th EuroSys Conference, EuroSys '20, Heraklion, Greece, April 27-30, 2020*. ACM, 2020, pp. 38:1–38:16. [Online]. Available: https://doi.org/10.1145/3342195.3387532

[65] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A. Sadeghi, and E. Stapf, "CURE: A Security Architecture with CUstomizable and Resilient Enclaves," in *30th USENIX Security Symposium, USENIX SEC '21, August 11-13, 2021*. USENIX Association, 2021, pp. 1073–1090. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/bahmani

[66] M. Kuhne, S. Volos, and S. Shinde, "Dorami: Privilege Separating Security Monitor on RISC-V TEEs," *CoRR*, vol. abs/2410.03653, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.03653

[67] V. Costan and S. Devadas, "Intel SGX Explained," *IACR Cryptol. ePrint Arch.*, p. 86, 2016. [Online]. Available: http://eprint.iacr.org/2016/086

[68] W. Zheng, Y. Wu, X. Wu, C. Feng, Y. Sui, X. Luo, and Y. Zhou, "A survey of Intel SGX and its applications," *Frontiers Comput. Sci.*, vol. 15, no. 3, p. 153808, 2021. [Online]. Available: https://doi.org/10.1007/s11704-019-9096-y

[69] C. Li, S. Lee, and L. Zhong, "Blindfold: Confidential Memory Management by Untrusted Operating System," in *32nd Annual Network and Distributed System Security Symposium, NDSS '25, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/blindfold-confidential-memory-management-by-untrusted-operating-system/

[70] 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 *15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17, Niagara Falls, NY, USA, June 19-23, 2017*. ACM, 2017, pp. 488–501. [Online]. Available: https://doi.org/10.1145/3081333.3081349

[71] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. S. Dwoskin, and D. R. K. Ports, "Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems," in *International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '08, Seattle, WA, USA, March 1-5, 2008*. ACM, 2008, pp. 2–13. [Online]. Available: https://doi.org/10.1145/1346281.1346284

[72] A. V. Hof and J. Nieh, "BlackBox: A Container Security Monitor for Protecting Containers on Untrusted Operating Systems," in *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI '22, Carlsbad, CA, USA, July 11-13, 2022*. USENIX Association, 2022, pp. 683–700. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/vant-hof

[73] F. Brasser, D. Gens, P. Jauernig, A. Sadeghi, and E. Stapf, "SANCTUARY: ARMing TrustZone with User-space Enclaves," in *Annual Network and Distributed System Security Symposium, NDSS '19, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/sanctuary-arming-trustzone-with-user-space-enclaves/

[74] Y. Xia, Z. Hua, Y. Yu, J. Gu, H. Chen, B. Zang, and H. Guan, "Colony: A Privileged Trusted Execution Environment With Extensibility," *IEEE Trans. Computers*, vol. 71, no. 2, pp. 479–492, 2022. [Online]. Available: https://doi.org/10.1109/TC.2021.3055293

[75] Y. Zhang, Y. Hu, Z. Ning, F. Zhang, X. Luo, H. Huang, S. Yan, and Z. He, "SHELTER: Extending Arm CCA with Isolation in User Space," in *32nd USENIX Security Symposium, USENIX SEC '23, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 2023, pp. 6257–6274. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/zhang-yiming

[76] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, "Komodo: Using verification to disentangle secure-enclave hardware from software," in *26th Symposium on Operating Systems Principles, SOSP '17, Shanghai, China, October 28-31, 2017*. ACM, 2017, pp. 287–305. [Online]. Available: https://doi.org/10.1145/3132747.3132782

[77] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang, "TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices," in *45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN '15, Rio de Janeiro, Brazil, June 22-25, 2015*.

IEEE Computer Society, 2015, pp. 367–378. [Online]. Available: https://doi.org/10.1109/DSN.2015.11

[78] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia, "Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 3, pp. 36:1–36:42, 2024. [Online]. Available: https://doi.org/10.1145/3700418

[79] P. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, "Intel TDX Demystified: A Top-Down Approach," *ACM Comput. Surv.*, vol. 56, no. 9, pp. 238:1–238:33, 2024. [Online]. Available: https://doi.org/10.1145/3652597

[80] R. Sahita, V. Shanbhogue, A. Bresticker, A. Khare, A. Patra, S. Ortiz, D. Reid, and R. Kanwal, "CoVE: Towards Confidential Computing on RISC-V Platforms," in *20th ACM International Conference on Computing Frontiers, CF '23, Bologna, Italy, May 9-11, 2023*. ACM, 2023, pp. 315–321. [Online]. Available: https://doi.org/10.1145/3587135.3592168

[81] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "TwinVisor: Hardware-isolated Confidential Virtual Machines for ARM," in *ACM SIGOPS 28th Symposium on Operating Systems Principles, SOSP '21, Virtual Event / Koblenz, Germany, October 26-29, 2021*. ACM, 2021, pp. 638–654. [Online]. Available: https://doi.org/10.1145/3477132.3483554

[82] S. Li, J. S. Koh, and J. Nieh, "Protecting Cloud Virtual Machines from Hypervisor and Host Operating System Exploits," in *28th USENIX Security Symposium, USENIX SEC '19, Santa Clara, CA, USA, August 14-16, 2019*. USENIX Association, 2019, pp. 1357–1374. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/li-shih-wei

[83] X. Xu, W. Wang, Y. Wu, Z. Min, Z. Pang, and Y. Jin, "virtCCA: Virtualized Arm Confidential Compute Architecture with TrustZone," *CoRR*, vol. abs/2306.11011, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2306.11011

[84] K. Onarlioglu, L. Bilge, A. Lanzi, D. Balzarotti, and E. Kirda, "G-Free: defeating return-oriented programming through gadget-less binaries," in *Annual Computer Security Applications Conference, ACSAC '10, Austin, Texas, USA, 6-10 December 2010*. ACM, 2010, pp. 49–58. [Online]. Available: https://doi.org/10.1145/1920261.1920269

[85] Y. Cheng, Z. Zhou, M. Yu, X. Ding, and R. H. Deng, "ROPecker: A Generic and Practical Approach For Defending Against ROP Attacks," in *21st Annual Network and Distributed System Security Symposium, NDSS '14, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014. [Online]. Available: https://www.ndss-symposium.org/ndss2014/ropecker-generic-and-practical-approach-defending-against-rop-attacks

[86] J. Hiser, A. Nguyen-Tuong, M. Co, M. Hall, and J. W. Davidson, "ILR: Where'd My Gadgets Go?" in *2012 IEEE Symposium on Security and Privacy, IEEE S&P '12, 21-23 May 2012, San Francisco, California, USA*. IEEE Computer Society, 2012, pp. 571–585. [Online]. Available: https://doi.org/10.1109/SP.2012.39

[87] J. Criswell, N. Dautenhahn, and V. S. Adve, "KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels," in *2014 IEEE Symposium on Security and Privacy, IEEE S&P '14, Berkeley, CA, USA, May 18-21, 2014*. IEEE Computer Society, 2014, pp. 292–307. [Online]. Available: https://doi.org/10.1109/SP.2014.26

[88] L. Davi, A. Sadeghi, D. Lehmann, and F. Monrose, "Stitching the Gadgets: On the Ineffectiveness of Coarse-Grained Control-Flow Integrity Protection," in *23rd USENIX Security Symposium, USENIX SEC '14, San Diego, CA, USA, August 20-22, 2014*. USENIX Association, 2014, pp. 401–416. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/davi

[89] N. Carlini and D. A. Wagner, "ROP is Still Dangerous: Breaking Modern Defenses," in *23rd USENIX Security Symposium, USENIX SEC '14, San Diego, CA, USA, August 20-22, 2014*. USENIX Association, 2014, pp. 385–399. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/carlini

[90] ARM Limited. (2025) OP-TEE Dispatcher. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware/blob/v2.13.0/services/spd/opteed/opteed_main.c

[91] ARM Limited. (2025) Trusty Dispatcher. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware/blob/v2.13.0/services/spd/trusty/trusty.c

[92] Julius Werner. (2021) QTI platform ports violate Binary Components policy? [Online]. Available: https://lists.trustedfirmware.org/archives/list/tf-a@lists.trustedfirmware.org/message/25UDM77JKUSMXATUKJVD5BTDFGHRKBYK/

[93] Google. (2025) Widevine. [Online]. Available: https://www.widevine.com/

[94] M. Busch, P. Mao, and M. Payer, "GlobalConfusion: TrustZone Trusted Application 0-Days by Design," in *33rd USENIX Security Symposium, USENIX SEC '24, Philadelphia, PA, USA, August 14-16, 2024*. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/busch-globalconfusion

[95] M. Busch, A. Machiry, C. Spensky, G. Vigna, C. Kruegel, and M. Payer, "TEEzz: Fuzzing Trusted Applications on COTS Android Devices," in *2023 IEEE Symposium on Security and Privacy, IEEE S&P '23, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 1204–1219. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179302

[96] S. Wan, M. Sun, K. Sun, N. Zhang, and X. He, "RusTEE: Developing Memory-Safe ARM TrustZone Applications," in *36th Annual Computer Security Applications Conference, ACSAC '20, Virtual Event / Austin, TX, USA, 7-11 December, 2020*. ACM, 2020, pp. 442–453. [Online]. Available: https://doi.org/10.1145/3427228.3427262

[97] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE," in *2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19, London, UK, November 11-15, 2019*. ACM, 2019, pp. 1723–1740. [Online]. Available: https://doi.org/10.1145/3319535.3363205

[98] N. Zhang, K. Sun, W. Lou, and Y. T. Hou, "CaSE: Cache-Assisted Secure Execution on ARM Processors," in *2016 IEEE Symposium on Security and Privacy, IEEE S&P '16, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016, pp. 72–90. [Online]. Available: https://doi.org/10.1109/SP.2016.13