How to Secure a RISC-V Embedded System in Just 30 Minutes

Donald Barnetson
Hex Five Security, Inc.
San Jose, CA, USA
don@hex-five.com

Cesare Garlati
prpl Foundation
Santa Clara, California, USA
cesare@prplfoundation.org

***Abstract*—The free and open RISC-V ISA defines many important building blocks of security. Properly implementing them is the system designer responsibility. So, the real question is: How does one properly secure a RISC-V embedded system? This paper offers a practical guide to using these security blocks to build a state-of-the-art Trusted Execution Environment (TEE) with a multitude of isolated security domains - Zones, and secure communications between them. The paper also shows how to verify Zone isolation and benchmark overall TEE system performance.**

Keywords—RISC-V; Embedded Security; Trusted Execution Environment; TEE; Secure boot; Root of Trust; Firmware

I. Introduction

Originally developed at U.C. Berkeley, the free and open RISC-V ISA promises to bring the innovation and collaboration of the open source community to the hardware world. When it comes to security, RISC-V specifications [1] provide many important building blocks and the rapidly growing RISC-V ecosystem even more. For designers used to traditional closed-source proprietary architectures, the complexity associated with properly implementing these new security technologies may prove daunting [2].

From a system design perspective, the real question is: How do I properly secure a RISC-V embedded system?

In this paper, we describe how to secure a RISC-V system using the free and open MultiZone Security Trusted Execution Environment (TEE) - developed and maintained by Hex Five Security, Inc. MultiZone Security provides signed boot, hardware enforced isolation for an unlimited number of security domains - Zones, a secure messaging system between Zones, secure interrupts, and operates on top of the standard RISC-V ISA. Throughout this paper we'll refer to the X300 a special version of the Rocket Core originally developed at U.C. Berkeley extended and maintained by Hex Five.

All software, bitstreams and documentation referenced in this paper are freely downloadable from the Hex Five open source repository on GitHub [3].

II. Trusted Execution Environment Primer

A TEE provides security through separation by running functional code blocks in isolated domains where their interaction with the system is defined by policies and enforced directly by the hardware.

A. Components of a modern trusted Execution Environment

A modern TEE is composed of a boot loader providing signed boot functionality from an immutable storage area - typically a mask ROM, and a microkernel operating at the highest level of privilege providing isolation, trap & emulation, safety and communications to separate blocks of user code running in physically isolated Zones [4].

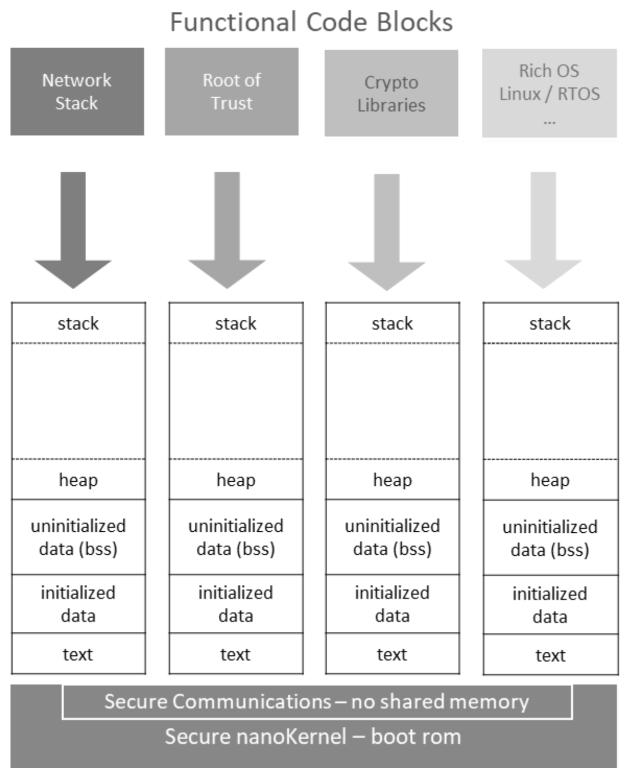

Fig. 1. – RISC-V MultiZone Trusted Execution Environment

The microkernel code should be as small and simple as possible to minimize attack surface and to allow formal verification.

B. Configuring a TEE

Legacy TEEs have suffered from a high level of complexity and the need for proprietary development and debugging processes [4]. This has been a cause of exploitable vulnerabilities creeping into these TEEs [5].

By contrast, more modern TEEs provide a simple and robust configuration process that assigns policies and ranges of memory mapped resources to each Zone. These include RAM, ROM, peripherals and interrupt sources. A configurator command line utility, fully integrated with the toolchain, verifies the security policies, merges together the fully linked binaries of each Zone, and generates the signed firmware image for upload [3].

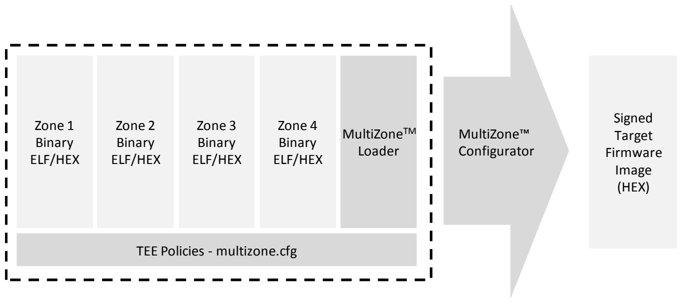

Fig. 2 – MultiZone Configurator flow: pre-compiled binaries, security policies and boot loader are combined into a single signed firmware image.

C. Debugging MultiZone Security

User code inside Zones can be debugged using industry standard tools such as gab, Opened and Eclipse IDE - the same way the unsecured discrete user code is.

Debugging can occur inside of a single Zone or can span multiple Zones [3].

III. SYSTEM SETUP

Bringing up an actual RISC-V system on an FPGA board provides a mechanism to evaluate the performance and resilience of the TEE:

A. FPGA Configuration

A Diligent Arty A7-T35 evaluation board is used with an Oilex ARM-USB-TINY-H Opened JTAG debugger. An open source version of the RISC-V Rocket Core is uploaded using the Xilinx Vivado toolchain.

All software is available at Hex Five's GitHub repository [3].

B. MultiZone TEE Configuration

The TEE is configured using a policy definition file named multizone that defines the preemptive scheduler tick and the mapping of Zones and resources:

1. Hardware interrupt sources are uniquely mapped to each Zone by listing them in the zone definition.
2. The first region of memory is for the text section – typically stored in flash or ROM. The program counter points to its base address when the Zone starts. Base address and size can be any multiple of 4 bytes. Flash or ROM would typically have [r]ear and e[x]acute privileges only.
3. The second region of memory is for the data segment – typically stored in RAM. Base address and size can be any multiple of 4 bytes. RAM would typically have [r]ear and [w]rite privileges only.
4. The remaining four regions are intended for generic memory mapped resources such as peripherals and need to be either naturally aligned power of two (NAPOT) or 4-byte length (4-BL) format. The privileges granted depend on the peripheral: for example, a real-time clock may have [r]ear only privileges especially if shared across multiple Zones.

C. User Mode Zone Binaries

The Zone binaries operate just like traditional executables, except they run in a secure user mode (lower privilege) rather than machine mode (highest privilege). The microkernel traps and emulates specific machine mode instructions to enable unmodified bare metal programs to run securely into a Zone without modifying the source code [3].

Optionally, the designer may want to link the open standard MultiZone C library to access optimized privileged instructions and additional microkernel runtime services - such as messaging and exception handling.

Specifically, the following services are utilized in this example and included in the file libhexfive.h:

- ECALL_YIELD – immediately yields execution to the next Zone rather than waiting for scheduler preemption.
- ECALL_SEND – send a fixed-size message (byte stream) to any Zone.
- ECALL_RECV – read the inbox for messages sent from any Zones.
- ECALL_TRP_VECT – registers a user mode handler against a trap. Each zone has a full set of trap / handlers for all synchronous traps.
- ECALL_IRQ – registers a user mode handler against an interrupt source. Interrupt sources include Platform Level Interrupts (Inter-core PLIC) and core-local interrupt (CLINT). Only one zone can be assigned to any asynchronous interrupt source.
- ECALL_CSRS/CSRC_MIE – atomically enables or disables all interrupts for a Zone.
- ECALL_CSR_XXX – read only secure emulation of privileged CSR instructions.

D. Demo Application Concept

In this configuration four Zones are configured as follows:

- Zone 1 – FreeRTOS running three tasks: CLI, Realtime Robotics and real-time fading LED – no TCP/IP stack attack surface.
- Zone 2 – PicoTCP and WolfSSL providing TCP/IP stack and TLS termination for the ethernet port – exposed to Internet attack and separated from RTOS.
- Zone 3 – Root of Trust for encrypted communications – fully isolated from other Zones.
- Zone 4 – USB bare metal terminal application to verify Zone separation and measure TEE performance.

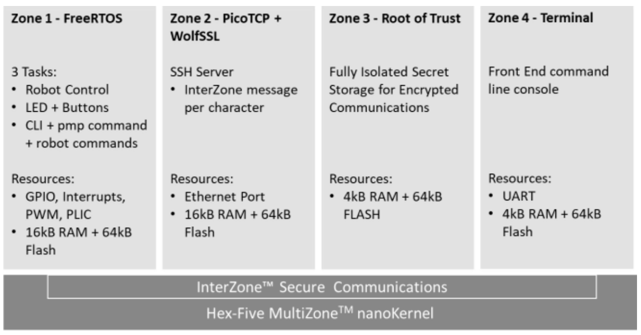
Zone 1 - FreeRTOS
3 Tasks:
• Robot Control
• LED + Buttons
• CLI + pmp command + robot commands
Resources:
• GPIO, Interrupts, PWM, PLIC
• 16kB RAM + 64kB Flash
Zone 2 - PicoTCP + WolfSSL
SSH Server
• InterZone message per character
Resources:
• Ethernet Port
• 16kB RAM + 64kB Flash
Zone 3 - Root of Trust
Fully Isolated Secret Storage for Encrypted Communications
Resources:
• 4kB RAM + 64kB FLASH
Zone 4 - Terminal
Front End command line console
Resources:
• UART
• 4kB RAM + 64kB Flash
InterZone™ Secure Communications
Hex-Five MultiZone™ nanoKernel

Fig. 3 – Demo Application Concept with four Zones.

IV. Installing, Testing and Debugging the MultiZone Security Demo

Installing and debugging MultiZone Security requires RISC-V standard open source development tools:

A. Download and Install the X300 FPGA Bitstream

Download the X300 bitstream from Hex Five's X300 repository and upload it to the Arty board using Xilinx's Vivado utility [6] and a micro USB connector.

B. Download and Install the RISC-V Toolchain

This can be done in many ways from different sources. In this example we download and install the GNU toolchain as indicated in the Hex Five's getting started guide [3]. You can either download binaries or recompile the full toolchain as you desire; Hex Five recommends they carefully crafted and thoroughly tested reference version of the RISC-V toolchain to ensure a proper implementation [3].

C. Download and Build MultiZone Security

Download the MultiZone Security SDK from Hex Five's open source repository on GitHub. The automated Make build process includes the following steps:

1. Each Zone is built into individual ELF files.
2. The MultiZone Configurator utility is invoked: it configures the nanokernel according to the content of the multizone.cfg file, it merges Zone binaries and nanokernel, and it signs the resulting target firmware image.

D. Upload the Signed Binary to the Arty Board

Using OpenOCD and the Olimex ARM-USB-TINY-H JTAG adapter – jumpers required, upload the signed binary to the Arty board by following the steps shown in the Hex Five getting started guide [3].

E. Demo Operation

The demo is operated by connecting a PC to the Ethernet and UART over USB ports of the Arty board.

Using a Telnet client of choice, connect the Ethernet to the FreeRTOS tasks running in Zone 1.

Using a serial terminal of choice, connect the UART over USB to the serial terminal console running in Zone 4.

The suggested evaluation flow includes:

1. Setup a console to ping the Arty board.
2. Deploy the robot and set it to operate through its loop command – send messages '>' to deploy and '1' to start the loop.
3. Measure context switch performance using "yield" and "stats" commands in Zone 4 under a variety of load conditions.
4. Reboot Zone 4 by sending a restart command to verify no impact on the other Zones – safety test.
5. Press BTN0-2 to cause an interrupt, changing LED color and generating messages to Zone 1 and 4.

All these real-time actions occur simultaneously without sharing any memory and only rely on the secure messaging infrastructure provided by the TEE.

Multiple debug breakpoints may be set across the four Zones and accessed simultaneously using standard gdb / OpenOCD tools.

V. Changing MultiZone Configuration

The system configuration can be modified to exercise different capabilities of the TEE. For example:

- Additional peripherals may be added to a Zone or moved from one zone to another.
- New tasks may be defined and added to FreeRTOS.
- The tick time may be changed in the multizone.cfg, or pre-emptive multitasking can be disabled by changing the tick time to 0 allowing context switching to only occur on a yield command.

If individual Zones are not needed as part of the demo, they can be simply excluded by deleting or commenting out their definition in the multizone.cfg file or by replacing the main() function with an infinite yield() loop - at least one Zone must be defined.

For those so inclined, the integrity of the signed boot process may be tested by modifying portions of the nanokernel in the signed binary prior to uploading it to the FPGA flash memory. In this case, the signature verification in the bootloader will fail and the boot process will be halted.

VI. CONCLUSIONS

The paper showed how to properly secure a RISC-V embedded system with a free and open Trusted Execution Environment providing signed boot and 4 Zones of isolation. A rich operating system inside a Zone is demonstrated with its TCP/IP and TLS stack secured by operating it out of an adjacent Zone and managing all communication via the secure messaging infrastructure provided by the TEE.

Areas for further research may include the use of split buffers to achieve faster TCP/IP communications, integration with rich operating systems – such as Linux, adding additional communications interfaces – such as USB, and other business functionality in the form of additional Zones.

ACKNOWLEDGMENTS

The authors would like to thank Sandro Pinto, Boran Car and Jose Martins for their contributions to this paper and the underlying software development that it documents. In addition, we would like to acknowledge SiFive, the maintainer of the open source E300 project that we forked into the X300 to extend its capabilities.

REFERENCES

[1] RISC-V Foundation. "RISC-V Specifications". [Online]: https://riscv.org/specifications/

[2] Cesare Garlati and Jack Greenbaum, "Hypervisors in Embedded Systems". Embedded World Conference 2018. [Online]: https://bringyourownit.com/2018/04/15/hypervisors-in-embedded-systems-applications-and-architectures/

[3] Hex Five Security Inc. "MultiZone Security", Git Repository, 2019. [Online]: https://github.com/hex-five/multizone-sdk

[4] Secure Technology Alliance "Trusted Execution Environment (TEE) 101: A Primer " June, 2018. [Online]: https://www.securetechalliance.org /wp-content/uploads/TEE-101-White-Paper-V1.1-FINAL-June-2018.pdf

[5] Sandro Pinto and Nuno Santos, "Demystifying Arm TrustZone: A Comprehensive Survey." ACM Computing Surveys, vol. 51, no. 6, article 130, December 2018.

[6] Di Shen. "Exploiting TrustZone on Android." Black Hat USA, 2015.

[7] Xilinx, Inc. "Xilinx Vivado Design Suite." 2019. [Online]: https://www.xilinx.com/products/design-tools/vivado.html