A Clean Slate Approach to Linux Security RISC-V Enclaves

Cesare Garlati
Hex Five Security
Redwood City, CA, USA
cesare.garlati@hex-five.com

Sandro Pinto
Universidade do Minho
Guimarães, Portugal
sandro.pinto@dei.uminho.pt

***Abstract* - Hardware consolidation requirements and sophisticated new functional requirements are forcing embedded systems designers to mix safety-critical applications with complex rich operating systems. The resulting mixed-criticality systems present orders of magnitude larger code base and unacceptably greater attack surface and system vulnerability – often exposed to remote attack. To address this emerging threat model, we propose a new zero-trust computing architecture based on the concept of multi zone enclaves for RISC-V based Linux systems.**

Keywords — RISC-V, Linux, Security, Separation, TEE, enclave, mixed-criticality, safety-critical, embedded, firmware.

I. INTRODUCTION

We live in the era of the Internet-of-Things (IoT). Billions of interconnected devices are now integral part of our lives, perform a myriad of functions, manage safety-critical operations, and generate and process vast amounts of sensitive data. As these systems are connected to the external world, they are inherently exposed to an endless number of cybersecurity threads [1, 2]. As shown by many recent high-profile cybersecurity incidents, the viability of this new Internet era heavily depends on the security of these devices and on the trust we are willing to put on them [2].

At its core, the ongoing cat-and-mouse game of exploits and patches is largely due to the intrinsic lack of security built into the prevailing computing model we all learned in school [1]. Mainstream operating systems (OSes) such as Linux and its many derivatives are built on a monolithic architecture, where most of the services have indiscriminate privileged execution rights [3]. A typical example is the Linux implementation of the TCP/IP stack, which is integral part of the kernel and, by definition, exposed to remote attack – the most damaging category of cyber threats. Although most of these systems rely on virtual memory to provide some level of separation, this widespread technique present significant limitations: (i) it requires complex dedicated hardware such as a Memory Management Unit (MMU), (ii) it introduces a large software attack surface necessary to drive the MMU – Linux kernel is approximately 18 million lines of code and growing, and (iii) it only provides separation between kernel services and user-space applications. This very architecture makes the kernel the single point of failure and the main avenue for attack via privilege escalation and via the inevitably high number of vulnerabilities exposed by such gigantic attack surface [1, 3].

In the context of secure computing systems, security through separation is a well understood principle traditionally implemented by Trusted Execution Environments (TEE) and/or Hypervisor software. TEEs have historically been used to statically partition hardware resources into two – a “secure world” for secrets and critical functionality, and a “non-secure world” for everything else including the rich operating system and its userland applications - i.e. Arm TrustZone [4]. Similarly, Hypervisor software leverages a 2-stage MMU to virtualize memory spaces so that OS and its applications cannot accidentally or intentionally reach into each other’s code and data segments. Legacy TEE technology suffers from two well understood intrinsic limitations: (i) the limited real-world applicability of the antiquated “two-worlds” model and (ii) the overwhelming complexity of the software components necessary for a proper implementation, which according to industry experts often results in these technologies not being used at all. This also applies to Hypervisor software that introduces the vast attack surface typical of the large code base necessary to operate MMU and hardware support for virtualization. In addition, over the last few decades a number of security researchers have questioned the very security of these technologies as their work resulted in the systematic discovery of many critical vulnerabilities [4, 5].

On the same note, recent research has demonstrated how the very high-performance architecture (e.g., caches, branch prediction, out-of-order execution) of modern processors can be easily exploited to conduct timing side-channel attacks as well as to interfere with determinism, predictability, and ultimately with system reliability [5, 6]. Timing side-channels can be used to compromise data confidentiality, which might be exploited to access private or sensitive data of either a TEE or the Hypervisor [5]. In the context of mixed-criticality systems (MCSs), the multiplexing of micro-architectural shared resources can lead to reciprocal interference and contention among virtual machines (VMs), which can break truly temporal isolation and hinder determinism by increasing jitter [6].

We start by presenting the standard security primitives specified by the RISC-V ISA. We then describe why legacy TEE technology is unsuitable for modern embedded applications and explain the main components required to meet these new requirements. Finally, we propose a new multi zone enclave for RISC-V based Linux systems targeting the Microchip PolarFire® FPGA and the details of its implementation.

II. RISC-V Primer

Recent advances in computer architectures have brought to light an innovative instruction set architecture named RISC-V, initially developed at U.C Berkeley and now ratified by the RISC-V Foundation. RISC-V differs from other computer architectures by offering a free and open instruction set architecture (ISA). RISC-V has the potential to become a security game-changer by defining a comprehensive set of security building blocks in the ISA itself, which are then available across the board in every silicon implementation. Some of these features include (i) privileged execution levels, (ii) physical memory protection (PMP), and (iii) user-mode interrupt delegation.

Level	Encoding	Name	Abbreviation
0	00	User / Application	U
1	01	Supervisor	S
2	10	Hypervisor	H
3	11	Machine	M

RISC-V Privilege Levels. The first security primitive offered by any RISC-V core is privilege levels. At any time, a RISC-V hardware thread (so-called hart), runs at a specific privilege level (Fig. 1). According to the current RISC-V Privileged ISA Specification (version 1.11) these include User/Application (U), Supervisor (S), and Machine (M). A fourth privilege mode, Hypervisor (H), is reserved and under review for ratification. Roughly speaking, these privilege levels are used to protect the different components of the software stack from each other. A synchronous non-maskable exception (trap) is raised if an instruction would result in the hart performing operations not permitted by the current privilege level.

RISC-V Physical Memory Protection (PMP). A second security feature is the PMP filter. It allows the highest privilege level (M) to protect specific memory regions and to grant lower privilege level access only to specific contiguous memory regions according to read / write / execute policies. This allows the partition of functionality between execution environments and other functional component behavior.

RISC-V User-Level Interrupts. A third relevant feature is the support for user-level interrupts - “N” extension. This extension allows the interrupt controller to delegate exception handling to more secure user-level handlers, bypassing the highest privilege level in the outer execution environment.

III. TEE Requirements For Modern Applications

As requirements for embedded and IoT systems have become more complex, system designers increasingly rely on 3rd party software components. These are typically available in the form of software libraries. In some cases, the source code is available for analysis and security validation – i.e. open source software, in some cases it is available as opaque linkable object modules, in some cases the source code is proprietary and completely hidden from scrutiny – i.e. binaries. Therefore, the requirement has emerged to enforce the separation, or firewalling, of the various software components within the system. However, the intrinsic complexity, code size, and attack surface of a rich operating system are simply unfit for the level of trust required in safety-critical applications. Imagine, for example, a simple embedded connected system that needs a TCP/IP stack, crypto libraries, Bluetooth stack, and a root of trust for secure boot and TLS operations. The complexity of each of these functional blocks leaves the system designer no other choice but to rely on a mix of more or less trusted 3rd party libraries [1].

In a traditional TEE architecture, secrets, cryptographic keys, certificates, and the libraries related to their functionality are combined into the single “secure world” available. Inside this world, each component is exposed and needs to trust all other components, de-facto breaking the zero-trust model [4, 5]. A single faulty instruction or software defect, intentional or unintentional, in one of these components has the potential to compromise them all [1, 3]. The second issue is complexity, which is the enemy of security. Embedded system developers are exposed to different programming models that are typically very different in a TEE than in a single world implementation. Commercial TEEs have TCBs of thousands of lines of code (and hundreds of KiB), with many unverified and untrusted external dependencies [5]. This is the main reason why TEEs are typically implemented only if required by the downstream customer. It also explains why commercial TEE implementations are regularly breached [4, 5].

We propose that a modern effective TEE must provide the following features:

- An extremely limited attack surface, in the order of few kilobytes. Assuming a conservative 1 defect per 1,000 lines of code ratio, a TCB equivalent to less than 1,000 lines of code.
- Completely self-contained with zero dependencies from libraries and other runtime components including C runtime, linker scripts, and kernel-mode drivers.
- Provided in the form of a sealed pre-built runtime, driven by statically defined user-defined policies, that doesn’t require or even expose to the developer any other interface than the policy configuration file itself.
- Isolation of executable code (text segments) to ensure that user programs run in unprivileged mode so that they can’t compromise the overall system integrity.
- Isolation of data (data segments) and memory-mapped peripherals (typically I/O) via a hardware unit that prevents access outside statically defined security boundaries.

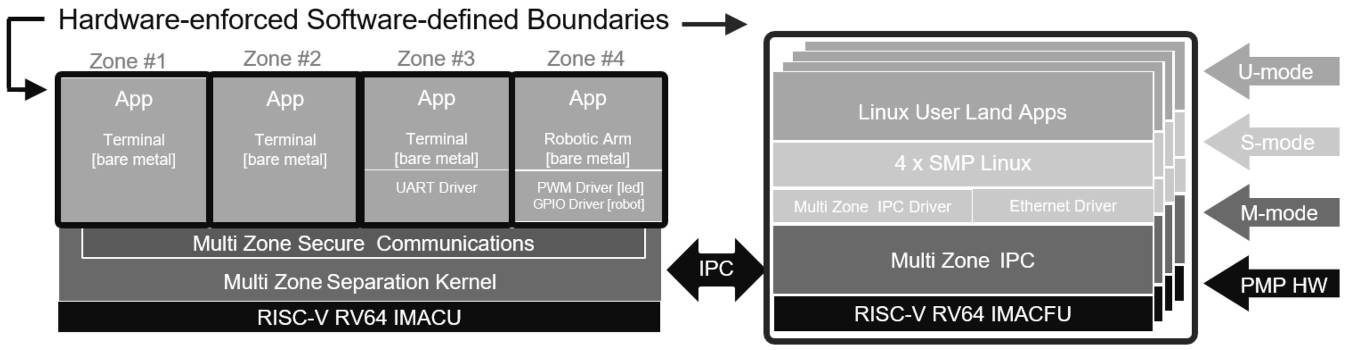


Fig. 2. Multi Zone Enclave For Linux - System Architecture

- Isolation of Interrupts so that interrupt handlers are mapped to the respective zone context and executed at a reduced level of privilege, unable to compromise the isolation model.
- Isolation of cores and other microarchitectural resources (e.g., caches) in heterogeneous systems such that low-criticality/non-secure core(s) cannot compromise and/or interfere with high-criticality/secure core(s).
- A preemptive temporal separation mechanism to ensure that any single Zone cannot cause a denial of service by indefinitely holding processing cycles, de facto taking the system to a halt. This is a must for safety-critical applications.
- A secure communications infrastructure to allow inter-zone data transfers without relying on shared memory resources such as buffers, stack, and heap.
- A secure inter-processor communications infrastructure to allow zones running on the secure core(s) to send/receive data to/from other low-criticality/non-secure core(s). In this case, shared buffers are inevitable and should be protected by exclusive allocation to the parties involved in the communication and by policies enforcing the respective consumer / producer role.
- A soft timer facility to multiplex the underlying hardware timer functionality to make it available to each zone independently from the others.
- Wait for interrupt functionality to allow transparent support for system suspend and low-power states.
- Trap and emulate functionality for privileged instructions to allow transparent robust implementation of existing application code originally designed to operate in a single unprotected memory space.
- An optional C language API to expose TEE runtime services such as messaging and process scheduling.
- A built-time command line utility for Linux, Windows, and Mac OS, fully integrated with toolchain and IDE, to combine and configure the zones binaries and to produce the signed firmware image for the secure boot of the system.

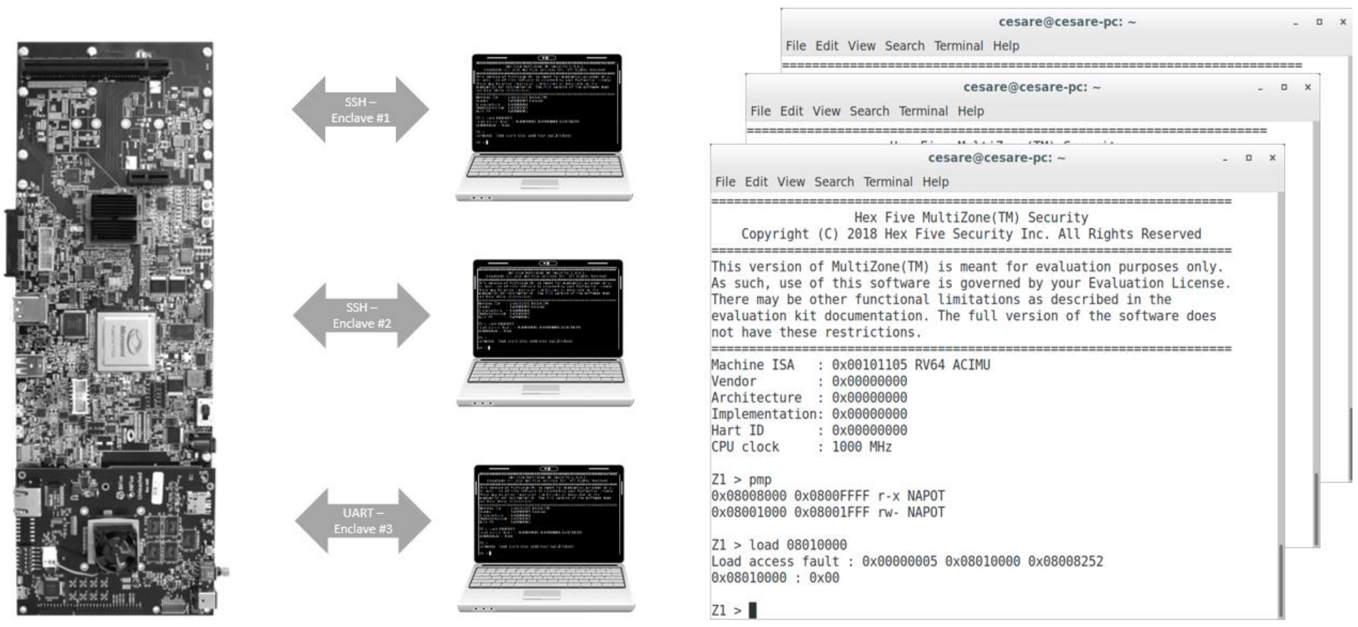
IV. MULTI ZONE ENCLAVES

Fig. 2 shows the system architecture of the proposed multi zone enclave for Linux [7]. This architecture is substantially different than traditional TEE architectures as it relies on a separated deterministic core to execute multiple trusted workloads alongside untrusted Linux applications.

Note: this architecture has been ideated by Hex Five for its MultiZone® Security TEE, the first commercial enclave specifically designed to bring security and separation to RISC-V based Linux systems – patent pending US 16/450,826 PCT US19/38774.

Multi Zone Secure Boot Process. The multi zone enclave implements a 2-stage secure boot loader to verify the integrity and authenticity of the firmware image - SHA-512. The standard Linux boot process has been modified with the addition of the multi zone secure boot loader (MSBL). This is responsible for securely loading and booting the trusted execution environment runtime, the user-mode enclaves binaries, and for handling the execution of the four Linux-capable cores to the subsequent boot stages: Zero Stage Boot Loader (ZSBL), First Stage Boot Loader (FSBL), and Berkeley Boot Loader (BBL). ZSBL, FSBL, and BBL boot stages are slightly modified to enforce overall system security policies and to prevent any possible promiscuity of untrusted Linux code with the secure enclaves.

Multi Zone Runtime. The multi zone runtime is comprised of two main components: (i) the preemptive separation kernel and (ii) the secure communications layer. The separation kernel provides time and spatial isolation across the multiple zones of the enclave. The kernel supports an unlimited number of isolated zones compliant to the policies defined in the configuration file by the system designer. Policies include read / write / execute access control and map the various physical resources of the system – i.e. RAM, ROM, I/O, - and interrupt sources to each zone (whitelist) and core (blacklist). The separation kernel implements a preemptive real-time scheduler with configurable round-robin and / or cooperative scheduling policies. In addition, it provides multiplexed timer events emulating one timer per each zone - in spite of the availability of a single architectural machine timer. Interrupts handling is fully isolated and emulated in the context of each zone. This includes low-latency vectored interrupts, preemptible



```
=====================================================================
              Hex Five MultiZone(TM) Security
   Copyright (C) 2018 Hex Five Security Inc. All Rights Reserved
=====================================================================
This version of MultiZone(TM) is meant for evaluation purposes only.
As such, use of this software is governed by your Evaluation License.
There may be other functional limitations as described in the
evaluation kit documentation. The full version of the software does
not have these restrictions.
=====================================================================
Machine ISA   : 0x00101105 RV64 ACIMU
Vendor        : 0x00000000
Architecture  : 0x00000000
Implementation: 0x00000000
Hart ID       : 0x00000000
CPU clock     : 1000 MHz

Z1 > pmp
0x08008000 0x0800FFFF r-x NAPOT
0x08001000 0x08001FFF rw- NAPOT

Z1 > load 08010000
Load access fault : 0x00000005 0x08010000 0x08008252
0x08010000 : 0x00

Z1 > █
```

Fig. 3. Multi Zone Enclave for Linux – Reference Application

interrupts, and Wait For Interrupt WFI (i.e., suspend mode). To provide complete and transparent support for unmodified binaries, the separation kernel handles trap & emulation for most protected instructions - i.e. CSR read-only.

Multi Zone Secure Communications. Zones inside the deterministic core communicate to each other via an exception handling mechanism immune to shared memory attacks. Communications across the cores rely on secure split buffers according to the multi zone open standard API AMP/Linux communication protocol. This protocol is designed for heterogeneous systems where real-time applications run in one or more MCUs alongside multiple AMP or SMP Linux-capable CPUs. The protocol is based on four invariants. These invariants specify the communication data path implemented through a protected shared memory split buffer: one half for zones 1 to n inboxes and one for the zone zero (Linux) inbox. These inboxes follow a statically defined layout sized according to the number of zones present in the system. The communication flow specifies that access to inboxes requires synchronization to avoid race conditions in multi-thread / multi-core systems.

A. *Reference Application*

Fig. 3 shows the reference application that we developed for system test and evaluation. It is designed to verify security and separation of a 4-zone enclave in a mixed-criticality system where Linux and real-time come together in a single RISC-V chip. The multi zone enclave is configured to run four separated bare metal applications (zones) fully protected from the Linux untrusted cores. Zone #1 and zone #2 connect to a PC terminal via SSH (Ethernet) to exercise and test secure connectivity, integration with the Linux IPC driver, enforcement of the isolation policies (PMP), performance statistics, soft-timer, interrupts, inter-processor communications. Zone #3 connects to a PC terminal via serial port (UART) to demonstrate peripheral mapping, secure user-mode drivers, enforcement of isolation policies, performance statistics, soft-timer, and inter-processor communications. Finally, Zone 4 operates a robotic arm connected via GPIO and blinks a heartbeat LED connected to the PWM hardware. Commands entered by the user via zone #1, #2, and #3 are dispatched via secure messaging to zone #4, which operates the robot's motors and reports back its status to the other zones.

B. *Security, Separation, And Performance Evaluation*

The reference implementation was evaluated on Microchip PolarFire® FPGA hardware. This system embeds five RISC-V cores: one deterministic microcontroller RV64 IMACU plus four RV64 IMACFU Linux-capable CPUs. In addition to extensive testing of security and separation, we benchmarked TCB size and performance metrics.

TCB Size. By design, the multi zone runtime is extremely small, written completely in assembly, and completely self-contained – no C libraries or runtime. Its TCB size is approximately 2KiB, which allows for formal verification.

Performance Overhead. To assess the performance overhead, we measured the zone context switch time. For the system under test, configured for four zones, a full context switch takes 260 clock cycles or 260ns at the operating frequency of 1GHz. If configured for a preemptive tick time of 10ms, the worst-case performance overhead is 0.0026%.

REFERENCES

[1] Chris Conlon and Cesare Garlati. “A New Zero-Trust Model for Securing Embedded Systems”. In Proceedings of the Embedded World Conference, Nuremberg, Germany, 2019.

[2] O. Alrawi, C. Lever, M. Antonakakis and F. Monrose, "SoK: Security Evaluation of Home-Based IoT Deployments." IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019.

[3] Simon Biggs, Damon Lee, and Gernot Heiser. "The Jury Is In: Monolithic OS Design Is Flawed: Microkernel-based Designs Improve Security." In Proceedings of the 9th Asia-Pacific Workshop on Systems, p. 16. ACM, 2018.

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

[5] David Cerdeira, Nuno Santos, Pedro Fonseca, Sandro Pinto. " SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems." IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA, 2020.

[6] M. Bechtel and H. Yun, "Denial-of-Service Attacks on Shared Cache in Multicore: Analysis and Prevention," 2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Montreal, QC, Canada, 2019, pp. 357-367.

[7] Hex Five Security Inc. “MultiZone® Security for Linux”. White Paper, CA, USA, 2019. [Online]: https://hex-five.com/wp-content/uploads/2020/01/multizone-linux-datasheet-20191216.pdf

[8] Sandro Pinto and Cesare Garlati. “User Mode Interrupts: A Must for Securing Embedded Systems”. In Proceedings of the Embedded World Conference, Nuremberg, Germany, 2019.