The industry-first secure IoT stack for RISC-V: a research project

Sandro Pinto and Jose Martins
Centro ALGORITMI - Universidade do Minho
{sandro.pinto, jose.martins}@dei.uminho.pt

***Abstract*—This paper describes the first fully functional framework for developing secure Internet-of-Things (IoT) systems on RISC-V processors. The proposed horizontal software stack is based exclusively on a number of open-source, commercial-grade technologies readily available today. The bedrock of the system is the Multizone Trusted Execution Environment (TEE), which provides the infrastructure to execute multiple isolated zones, communicating through a secure messaging infrastructure, and other security primitives such as a secure boot. The remaining components, each encapsulated in a different zone, include (i) a modified, secure implementation of FreeRTOS, with full support for user-mode interrupts, (ii) a TCP/IP stack (picoTCP) complemented by a TLS library (wolfSSL), (iii) a minimalist root of trust (RoT) implementation for key management, and (iv) a command-line interface for overall system management. All the developed components, including a modified version of SiFive's E300 platform, running on a Digilent ARTY 7 FPGA board, are open-sourced under an Apache-2.0 license and freely available on GitHub.**

***Index Terms*—Secure stack, RISC-V, IoT, TEE, open-source.**

I. Introduction

The modern technology landscape is finally arriving at the longtime prophesied Internet-of-Things (IoT), where devices from all ends of the spectrum, performing a myriad of functions, and many times, managing safety-critical operations and generating and handling vast amounts of sensitive data, are connected to the Internet. As such, this new reality is not only enriching our everyday lives but simultaneously creating several new risks as shown by recent cybersecurity incidents. For example, Mirai Botnet has clearly demonstrated that the success of this new Internet era is heavily dependent upon the trust and security built in these IoT devices [1].

In the last few decades, security through obscurity, the approach followed by a majority of industry players, has been proven time and time again to be ineffective. Therefore, as pointed out as early as 1975 by Saltzer and Schroeder [2], a paradigm shift towards open designs is a must. This principle, embodied today in the Free and Open-Source Software (FOSS) movement, stresses the fact that security mechanisms must not rely on the secrecy of their inner-workings, which realistically will not remain secret for widely deployed systems. Moreover, it allows the system to be inspected by many reviewers and makes it easier for a designer to directly evaluate whether the system fulfills the security requirements for the intended application. Although some would counterpoint that systems such as Linux cannot be trusted since, despite following the open-source approach, it still shows a large number of security vulnerabilities, this is a consequence of its large, complex and monolithic structure [3], not its open-source philosophy. Horizontal, microkernel-based architectures which adhere to the principles of minimality and of the least privilege, providing fine-grained encapsulation and fault-containment, are inherently more secure, and therefore essential for developing secure IoT systems.

In recent years, the open-source mindset has been making its way towards hardware development. This has culminated with the emergence of RISC-V which is experiencing rapid adoption by both academia and industry. RISC-V is a free instruction set architecture (ISA) based on the principles of simplicity and openness. By following these principles, RISC-V takes the first step to be a game-changer for hardware security. Moreover, some of the features provided by the architecture, besides the usual virtual memory and privilege level mechanisms, further enhance this security focus. These include, for example, the Physical Memory Protection (PMP) mechanism and the user-level interrupt N extensions, which are especially important to achieve thorough containerization in low-end microcontrollers [4].

Taking all the above considerations in mind, in this paper, we propose the first secure IoT stack for RISC-V systems, based solely on the use of open-source components executing on top of a minimalistic Trusted Execution Environment (TEE).

II. Secure IoT Stack

Fig. 1 depicts the implemented software stack for developing secure RISC-V IoT devices. The system runs on top of a modified SiFive E300 which implements a RISC-V 32-bit core with the machine and user modes (M- and U-mode) and relies on the Multizone TEE to provide the infrastructure to securely boot and execute multiple isolated zones. The system is configured to run four independent zones which implement the basic functional blocks typically present in embedded connected devices, such as smart sensor nodes and IoT endpoints in general. Zone 1 runs the industry-first RISC-V secure implementation of the popular FreeRTOS, zone 2 runs a secure TCP/IP stack, zone 3 provides a minimalist root of trust (RoT) for key management, and zone 4 runs a classic command-line interface (CLI) for overall system management. All zones are completely isolated and communicate through well-defined message-based interfaces provided by the secure InterZone Messenger. The full stack is based on open-source

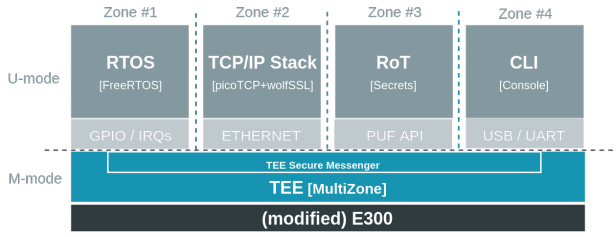

Fig. 1: Secure IoT Stack Architecture for RISC-V

technologies and all developed components are open-sourced under an Apache-2.0 license [5] - the remaining of this section provides a glance over each system component.

A. Hardware Platform

The hardware platform is based on the open-source SiFive Freedom E300 which, in turn, is based on a RISC-V 32-bit core. The system on a chip (SoC) targets the low-cost Arty FPGA board developed by Digilent. The Freedom E300 design was modified to include the Xilinx Ethernet Lite component necessary to operate the Arty on-board ethernet port. The SoC was also modified to add one-time programmable (OTP) storage for the TLS certificates. Other notable hardware enhancements include the addition of memory RAM up to 64 kB, the increase of the CPU frequency to 65 MHz, the addition of a second privileged level (user mode), the support for core-local interrupts, and the modification of the cache architecture (4-way associative).

B. MultiZone TEE

MultiZone Security is the first TEE designed from the ground up to leverage the hardware ”hooks” built into the standard RISC-V ISA. MultiZone Security provides signed boot and segregates the various functional blocks into an unlimited number of physically separated ”Zones”. The MultiZone nanokernel is a lightweight, formally verifiable, bare metal kernel providing policy-driven hardware-enforced separation of resources (e.g., RAM, ROM, I/O, and interrupts). Inter-zone communications are secured via the InterZone messenger, which uses no shared memory. Finally, with the MultiZone Configurator, the system designer defines read/write/execute policies and maps various physical resources to each Zone.

C. Secure FreeRTOS

FreeRTOS is a popular and widespread RTOS among academia and industry. Although enjoying widespread applicability, FreeRTOS has been struggling in its own security problems. To implement a secure version of FreeRTOS on top of MultiZone, we de-privileged the OS to userland. While MultiZone provides full support for trap and emulation (which inherently requires no modifications), modifications were mainly driven to address performance and security. Access to privileged instructions was replaced by explicit and direct calls (MultiZone API) to the nanokernel. Modifications were implemented across five different OS subsystems: startup code, task management, context switching, exception handling, and time management. Among these modifications, we highlight the implementation of full support for user-mode interrupts [4]. On top of FreeRTOS runs three main threads (or tasks in FreeRTOS terminology) which implement the application logic for controlling a robotic arm, fading and blinking LEDs, handling three buttons, and interfacing a CLI which is mainly used to control the robot through the Internet.

D. Secure TCP/IP stack

Zone 2 provides network connectivity by implementing a secure TCP/IP server. As the TCP/IP stack we opted for the picoTCP due to its small footprint and modular implementation; moreover, picoTCP is actively being maintained and the code is distributed under GPLv2 and GPLv3 license. To ensure secure communication over the network we have also added TLS through an embedded crypto library. For the crypto library, we opted for wolfSSL which is the de-facto open-source SSL/TLS library for resource-constrained environments. wolfSSL was configured to support the latest TLS protocol, i.e. the TLS v1.3.

E. Root of Trust

Zone 3 provides a minimalist implementation of a root of trust which is based on an OTP memory that permanently stores the certificates and keys to secure the TLS communication. By ensuring keys and certificates are isolated into a dedicated zone, it allows applying hardware-enforced policies to grant access to the certificates only from the TCP/IP server zone.

F. Command-line Interface

Zone 4 implements the front-end of a classic command-line interface. is accessed through a serial interface and aims at providing an easy channel to manage and benchmark the overall system’s functionalities and performance.

III. Conclusion

Open, secure-by-design approaches seem to be the most viable aisle for truly developing secure IoT devices. Furthermore, large monolithic operating systems have shown to be inappropriate to serve as the foundation of these systems. In this paper, we described our prototype of the first secure IoT stack for RISC-V systems, solely based on open-source technology. All the components execute on top of a microkernel-based TEE which takes advantage of hardware features provided by all RISC-V hardware.

References

[1] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, “DDoS in the IoT: Mirai and Other Botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, 2017.

[2] R. E. Smith, “A Contemporary Look at Saltzer and Schroeder’s 1975 Design Principles,” *IEEE Security & Privacy*, vol. 10, no. 6, pp. 20–25, Nov 2012.

[3] S. Biggs, D. Lee, and G. Heiser, “The Jury Is In: Monolithic OS Design Is Flawed,” in *9th Asia-Pacific Workshop on Systems*, 2018.

[4] S. Pinto and C. Garlati, “User Mode Interrupts: A Must For Securing Embedded Systems,” in *Embedded World Conference*, 2019.

[5] Hex-Five, “MultiZone Secure IoT Stack,” https://github.com/hex-five/multizone-secure-iot-stack/, 2019, [Online; accessed 20-Feb-2019].