Trusted Execution Environments

A System Design Perspective

Boran Car
Hex Five Security, Inc.
London, UK
boran@hex-five.com

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

***Abstract*—The Internet of Things (IoT) represents a collection of billions of smart, connected devices. Current approaches to securing IoT devices typically go through the addition of complex hardware mechanisms or the implementation of heavy containerization and virtualization solutions. In this paper, we take the reader through designing a real-world scenario of an IoT device making use of Trusted Execution Environments (TEE) to securely isolate different parts of the system. We aim to demonstrate a network connected device resembling a typical IoT device with a clear boundary separation between the application, the networking stack, and the root of trust.**

Keywords—MultiZone; security; firmware; Trusted Execution Environment; TEE; TCP/IP; Root of Trust; IoT

I. INTRODUCTION

The Internet of Things (IoT) field has proliferated, with current estimates at 11 billion devices according to a recent Development Bank of Singapore (DBS) report [13]. According to the same report, privacy, security, and interoperability are the key barriers for widespread adoption [11]. While attempts at interoperability and standardization exist with organizations such as EdgeX [9], privacy and security still remain largely unaddressed. IoT devices do not work in isolation, they typically need to communicate with a central manager, posting their results and accepting commands from that central manager. Typical forms of communication involve protocols like BLE and TCP/IP. These protocols bring complex serializers and de-serializers often vulnerable to buffer overflow exploits, use after free and so on, with the latest examples being vulnerable TCP/IP stacks that could be exploited [12].

Current solutions involve process isolation, sandboxing, and containerization offered by the operating system (OS). The idea is to limit the attacker's ability to pivot once a vulnerability is exploited. However, embedded devices are typically resource-constrained and have limited memory and processing power. Microcontroller-class devices usually don't have the hardware memory-management unit (MMU) needed to run a rich OS with containerization and sandboxing like Linux.

While microkernel-based solutions are also a valid approach security-wise, for their minimal privileged kernel code and process isolation, they also require physical MMU - F9 being the only identified exception [10, 11]. Therefore, embedded OSes running on hardware without the MMU tend to combine all functional code blocks in the same privilege space, which leads programs to share the same access to code and data. Without access control, a compromised component of the software stack can bring the entire system down. Solving the problem without privilege separation usually comes in the flavor of protecting the memory via managed languages or annotations. This is intended to prevent shared memory exploits like buffer overflows, underruns, use after free and similar. Such methods do not address post-exploitation threats and need a complementary new approach to security.

In this paper we look at that complementary, whole system design and partitioning approach, while striving to adopt the principle of least privilege for the partitions. We take the spirit of "Every program and every privileged user of the system should operate using the least amount of privilege necessary to complete the job" [14] and adapt it to partitions. We also take inspiration from the cloud design pattern, specifically the orthogonal security enhancement via a sidecar approach [3] and aim to provide an orthogonal, embedded friendly, opt-in approach for those connected devices that make up by the billions the Internet of Things.

II. A NEW RADICAL IDEA: MULTIZONING

A common occurrence in the cloud world is the sidecar pattern [4]. The sidecar pattern allows to isolate chunks of functionality into separate containers. The Istio Service Mesh [15] takes this model a step further and uses sidecars to provide mutual transport layer security (mTLS) communications between microservices - without the developers having to worry about certificates or policies. A key advantage is the orthogonal composability, where a developer can write locally, with the frameworks he is most familiar, with and opt for extra security or functionality at deployment time. We wanted to combine these advantages with the interprocess communications (IPC) model typical of the microkernel approach. The developer writes the application with the universal asynchronous receiver transmitter (UART) logic in mind and, with minimal changes, adds another deployment zone for mTLS. We also considered how vsftpd [7] partitions a single process into a set: vsftpd has separate processes handling different tasks, each of those processes running with the minimal privileges needed for the specific task [8].

As an initial demonstration, we propose a system of multiple zones in addition to the application zone. The zones communicate with each other via InterZone communications

(IZC), similar to processes communicating via IPC but without introducing the attack surface typical of monolithic kernels. One zone is dedicated to TLS termination and secure connectivity into the system. From the application zone's perspective, this zone is no different than a zone providing UART connectivity - they both implement the stream interface. The third zone provides the Root of Trust for secure storage of certificates and key management. In a typical cloud scenario - such as AWS IoT Greengrass or EdgeX - the application zone would be developed by the OEM, the TLS zone by the cloud service provider, and the Root of Trust zone by the device manufacturer.

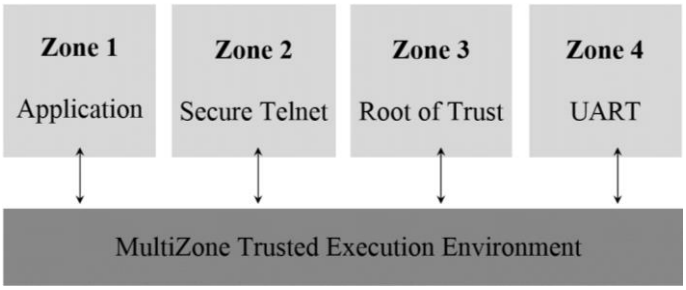

Fig. 1. MultiZone Security Configuration

III. PROOF OF CONCEPT

We based the design of our proof of concept on MultiZone Security [5], an open standard lightweight Trusted Execution Environment freely available on GitHub under the Apache 2.0 license. MultiZone Security is designed for RISC-V [1], the free and open Instruction Set Architecture (ISA) that is quickly disrupting the whole processor industry. Our hardware implementation is based on the Freedom E300, the open source System on a Chip (SoC) maintained by SiFive [2]. This SoC targets the low-cost Arty FPGA board developed by Digilent. Although the Freedom E300 SoC doesn't include ethernet capability, it is open-source and this allowed us to add the Xilinx Ethernet Lite component necessary to operate the on-board ethernet port. We opted for the Ethernet Lite component to avoid the complexity of direct memory access (DMA), which is beyond the scope of our security research and often not required by typical IoT edge nodes. We also modified the hardware to add one-time programmable (OTP) storage for the TLS certificates. The OTP data is written during the FPGA programming and can only be read afterwards. Other notable hardware enhancements we made include: adding more RAM up to 64 KiB, increasing the CPU frequency to 65 MHz, enabling RISC-V user mode, and adding support for core-local interrupts. Thanks to our sponsor Hex Five Security, and in the best spirit of true open source collaboration, we made all these enhancements freely available on GitHub for the greater good of the broader RISC-V community [4].

On the software side, we rely on the MultiZone Security Trusted Execution Environment to partition the hardware into four separate zones - as shown in Fig. 1. The Application, the Secure TCP/IP Server, the Root of Trust and the UART all get their separate zones and communicate according to a well-defined secure protocol. This protocol is based on the InterZone Messenger layer provided by the TEE with the addition of a simple schema to support message acknowledgements. MultiZone messages are fixed in size and organized in 16-byte long datagrams. Fig. 2 shows how we organized the four 4-byte blocks to implement a simple back-pressure algorithm. The sender sends a message to destination and waits for acknowledgement before sending a new message. The receiver acknowledges the data by replying to the message. Having separate ACK and MSG counters supports separate FIFO structures for producer and consumer and a configurable number of in-flight messages before applying back-pressure.

The application zone responds to the user commands, reports the status of the on-board push buttons connected to interrupts and controls in real-time the movements of a robotic arm connected via SPI. This is intended to demonstrate a typical Industrial IoT scenario including local sensing and actuation capabilities and remote control via a secure Internet link. Initially, we wrote and tested the standalone application using only UART connectivity. Then we "dropped" the fully linked executable into the MultiZone TEE to decouple its safety-critical control logic from the Internet connectivity zone exposed to remote attack.

For the Internet-exposed zone, we chose the picoTCP [16] TCP/IP stack because it is open source, light weight and highly configurable. For the TLS library we opted for wolfSSL, which is also open source and easy to combine with picoTCP. In addition, we developed a simple telnet server to set up the link and forward data to and from the application zone.

Our minimalist implementation of the root of trust is based on a One Time Programmable (OTP) memory that permanently stores the certificates necessary to secure the TLS communications. Having it isolated in a separate zone allows to apply hardware-enforced policies to grant access to the certificates only from the TCP/IP Server Zone.

The UART zone offers an independent local side-band channel to verify and benchmark the functionality of the MultiZone Trusted Execution Environment via a simple user console.

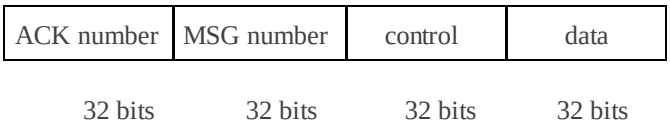

Fig. 2. InterZone message synchronization protocol

IV. CONCLUSION

Our work showed how to secure a typical IoT application into a set of independent, physically isolated functional blocks – Zones – without the cost and complexity typical of antiquated proprietary primitives. We leveraged a complete set of innovative open source technologies including RISC-V processors, MultiZone Security TEE, picoTCP, wolfSSL and FreeRTOS. In the best spirit of open source development, we have made all our work freely available on GitHub – see reference to Hex Five repository [4].

Areas for future development may include adding OpenSSH access, support for crypto accelerators, and a more performant implementation of the inter-zone message synchronization protocol. In addition, UART, Root of Trust, and TCP/IP zones could be repackaged as reusable MultiZone components, similar to Docker containers for silicon.

V. REFERENCES

[1] RISC-V Foundation. “RISC-V Instruction Set Manual Volume I: User Level ISA Document Version 2.2” https://riscv.org/specifications/

[2] SiFive, Inc. “E300 RISC-V Respository” https://github.com/sifive/freedom

[3] ThoughtWorks Technology Advisory Board. Technology Radar. Tech. rep. Nov. 2017, p. 10.

[4] Hex Five Security, Inc. hex-five/multizone-fpga. 2019 https://github.com/ hex-five/multizone-fpga.

[5] Don Barnetson Cesare Garlati. Hex Five - MultiZoneTM Security. http://hex-five.com

[6] Microsoft Corporation. Sidecar pattern. 2017. https://docs/microsoft.com/en-us/azure/architecture/patterns/sidecar

[7] Chris Evans. vsftpd. https://security.appspot.com/vsftpd. html

[8] Chris Evans. vsftpd Design. https://security.appspot.com/ vsftpd/DESIGN.txt

[9] EdgeX Foundry. EdgeX Project Introduction. 2018. https://www. edgexfoundry.org/wp- content/uploads/sites/25/2017/12/ EdgeX-Introduction-Full-12.28.17.pdf

[10] Jim Huang. F9: A Secure and Efficient Microkernel Built for Deeply Em- bedded Systems. 2013. https://www.slideshare.net/jserv/ f9-microkernel

[11] Jim Huang and Louie Lu. Secure Microkernel for Deeply Embedded De- vices. 2017. https://archive.fosdem.org

[12] Ori Karliner. FreeRTOS TCP/IP Stack Vulnerabilities Put A Wide Range of Devices at Risk of Compromise: From Smart Homes to Critical Infrastruc- ture Systems. 2018. https://blog.zimperium.com/freertos- tcpip - stack - vulnerabilities - put - wide - range - devices - risk - compromise - smart - homes - critical - infrastructure - systems

[13] Sachin Mittal, Tsz Wang Tam, and Chris Ko. Internet of Things The Pillar of Artificial Intelligence. Tech. rep. 63. June 2018.

[14] Jerome H. Saltzer. “Protection and the Control of Information Sharing in Multics”. In: Commun. ACM 17.7 (July 1974), pp. 388–402. ISSN: 0001- 0782. DOI: 10.1145/361011.361067. http://doi.acm.org/ 10.1145/361011.361067.

[15] Jimmy Song. Understanding How Envoy Sidecar Intercept and Route Traf- fic in Istio Service Mesh. https://medium.com/devopslinks/ understanding- how - envoy- sidecar- intercept- and- route- traffic - in - istio - service - mesh - 20fea2a78833 (visited on 01/10/2019).

[16] Altran Intelligent Systems. picoTCP Wifi. 2017. https://github. com/tass-belgium/picotcp/wiki