Cybershield: Secure Boot for Obfuscated Instruction Codes

Garrett R. Perkins*, Tristan Running Crane†, Hezekiah A. Austin†, Benjamin Macht†, Chris Major ¶, Ann Marie Reinhold*‡, Clemente Izurieta*‡§, and Brock LaMeres†¶

*Gianforte School of Computing, Montana State University, Bozeman, MT, USA
†Electrical & Computer Engineering Department, Montana State University, Bozeman, MT, USA
‡Idaho National Laboratory, Idaho Falls, ID, USA
§Pacific Northwest National Laboratory, Richland, WA, USA
¶Resilient Computing, Bozeman, MT, USA

***Abstract*—The increasing reliance on Field Programmable Gate Arrays (FPGAs) in security-critical applications underscores the need for robust protection mechanisms against cyber threats such as buffer overflow and injection attacks. This paper presents Cybershield, a novel integration of a Trusted Execution Environment (TEE) within RadPC, a radiation-tolerant softcore processor featuring Quad Modular Redundancy (QMR). Leveraging RadPC's four-core architecture, Cybershield's TEE employs secure boot from non-volatile memory to initialize RadPC's cores with obfuscated instruction codes. The opcode obfuscation and a hardware-based anti-voter mechanism prevent the execution of unauthorized code and detect Indicators of Compromise (IoCs). By implementing secure boot and opcode obfuscation, Cybershield mitigates common attack vectors while maintaining software redundancy and recoverability. Experimental validation demonstrates the system's ability to detect buffer overflow attacks and prevent unauthorized code execution. While the integration of a TEE introduces computational overhead and development constraints due to RadPC's bare-metal environment, this work lays the foundation for combining hardware and software redundancy to enhance the security of embedded systems.**

***Index Terms*—Trusted Execution Environment (TEE), Obfuscation, Quad Modular Redundancy (QMR), Field Programmable Gate Array (FPGA), RISC-V**

I. Introduction

The rapid growth of the Internet of Things (IoT) and edge computing has fundamentally transformed the way data is processed and analyzed. Unlike traditional cloud computing, edge computing shifts data processing closer to the source, enabling real-time insights, reduced latency, and minimized bandwidth usage. This paradigm shift has fueled advancements in a wide range of industries, from healthcare and automotive systems to industrial automation and telecommunications. However, as these edge devices become increasingly interconnected and intelligent, their attack surfaces expand, exposing them to a myriad of cybersecurity threats.

At the heart of many edge devices are Field Programmable Gate Arrays (FPGAs), which offer unparalleled flexibility, reconfigurability, and performance [1]. FPGAs combine programmable logic with embedded processors, and have become particularly attractive for edge computing due to their ability to handle complex workloads such as artificial intelligence (AI), signal processing, and real-time control. Despite their advantages, FPGAs were not originally designed with robust security in mind. As they transition from specialized applications to widespread use in critical industries including but not limited to radar, Unmanned Aerial Vehicles (UAVs), Industrial Control Systems (ICS), data centers, neural networks, and space avionics, the potential for malicious attacks targeting both their hardware and software components has grown significantly [2], [3].

This increasing reliance on FPGAs in security-critical applications highlights the urgent need to address their vulnerabilities. From intellectual property theft to fault injection and side-channel attacks, FPGAs face a diverse array of threats [4], [5]. To ensure the reliability and security of edge computing systems, new mechanisms must be developed to protect the hardware, software, and communication protocols of FPGAs.

Trusted Execution Environments (TEEs) offer a promising solution to enhance the security of FPGAs. TEEs come in many forms with many features. Most major CPU vendors have introduced their own chip-specific TEEs. These TEEs, i.e. *ARM TrustZone*[1,2], *Intel SGX*[3], and *AMD SEV*[4] leverage hardware-based security features to protect against vulnerabilities that traditional process isolation methods cannot address [6]. By isolating sensitive operations and providing secure boot mechanisms, TEEs can protect edge devices from tampering and unauthorized access. This paper presents the novel design and integration of a TEE with a radiation-tolerant softcore processor, RadPC, implemented on a commercial FPGA platform. This integration is known as CyberShield.

II. Related Works

Trust Execution Environments specific to FPGAs come in many different forms. Some TEEs host a myriad of features

[1]https://www.arm.com/technologies/trustzone-for-cortex-a
[2]https://www.arm.com/technologies/trustzone-for-cortex-m
[3]https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html
[4]https://www.amd.com/en/developer/sev.html

and have more general applications, whereas others are use-specific and intended for a single purpose [7].

Feature-rich TEEs aim to provide comprehensive security solutions by integrating elements such as memory encryption, secure boot, attestation, and root of trust, making them suitable for multi-tenant environments and dynamic workloads [7]. Conversely, application-specific TEEs are tailored to address distinct security challenges, such as mitigating side-channel attacks or securing intellectual property within FPGA deployments [7].

Secure boot is a fundamental security feature in TEEs, ensuring that the system initializes in a trusted state by verifying the authenticity and integrity of critical components. Several works have explored different aspects of secure boot in TEEs within FPGAs.

Recent advancements in secure boot architectures have explored the integration of hardware-based security measures to improve the efficiency and robustness of firmware verification. Loo et al. [8] propose a secure boot implementation for RISC-V using an FPGA-based approach. Their method introduces a hardware security block at the Register Transfer Level (RTL) to generate a SHA-512 digest, significantly reducing the computational burden on software-based secure boot mechanisms. The study highlights that while secure firmware incurs a 35% increase in boot time and a 3.3 MB increase in binary size, the FPGA implementation offsets performance costs by accelerating cryptographic computations, achieving an 1132% improvement in execution time over software-based hashing.

Another advancement in secure boot architectures is CARE, a lightweight attack-resilient secure boot framework designed for RISC-V-based SoCs [9]. Unlike traditional secure boot mechanisms that focus solely on detecting malware presence, CARE introduces an onboard recovery mechanism that ensures compromised devices can autonomously restore their firmware to a trusted state. The framework integrates a Code Authentication and Resilience Engine (CARE) that verifies firmware integrity using a hardware-accelerated HMAC-SHA256 cryptographic core. In the event of a detected compromise, CARE employs a dedicated recovery engine to re-flash only the corrupted memory regions from a secure backup ROM, preventing unauthorized modifications and minimizing system downtime. By leveraging Physical Memory Protection (PMP) and secureIbex features of the RISC-V processor, CARE mitigates fault injection and side-channel attacks while maintaining a minimal 8% performance overhead.

One of the most notable RISC-v-based TEEs, Keystone, incorporates secure boot as a critical security primitive, ensuring that only authenticated and unmodified software is executed within its TEE [10]. The secure boot process in Keystone begins with a hardware-based root of trust, where a tamper-proof bootloader or cryptographic engine measures the integrity of the Security Monitor (SM) at system reset. This process generates a fresh attestation key, which is securely stored in the SM's isolated memory and signed using a hardware-visible secret. During boot, Keystone verifies the integrity of enclave code and runtime components before execution, preventing unauthorized modifications or tampering. By leveraging RISC-V's Physical Memory Protection (PMP) and a minimal trusted computing base (TCB), Keystone ensures strong memory isolation and attestation capabilities, making it adaptable for various deployment scenarios, from cloud environments to embedded systems.

Another notable approach by Streit et al. focuses on securely booting from non-volatile memory (NVM) in insecure environments by leveraging the reconfigurable logic of an FPGA as a secure anchor point. The proposed methodology integrates a Trusted Memory-Interface Unit within the FPGA's reconfigurable logic region, enabling integrity and authenticity verification of NVM data before executing any user applications. The boot image is decrypted using the dynamically generated encryption key, and its integrity is verified by comparing the calculated hash against the stored token [11].

III. Background

A. RadPC

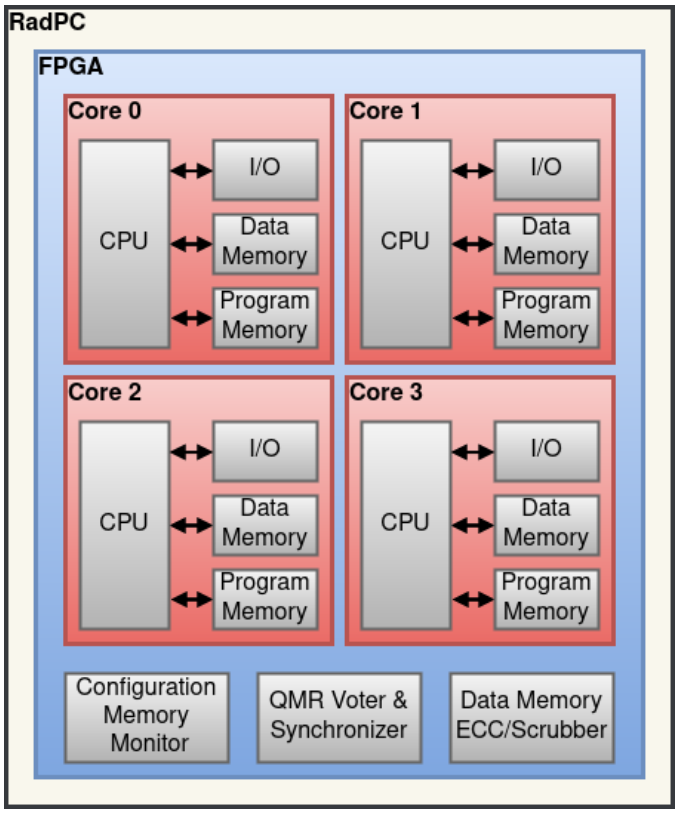

Fig. 1: Block diagram of the QMR architecture of the RadPC FPGA system [12].

The softcore processor RadPC was chosen for integration with a TEE. RadPC is a custom radiation-tolerant space computer designed for small satellites with a reduced instruction set (RISC-V 32I). The radiation tolerance of RadPC specifically relates to Single Event Effects (SEEs), which are strikes that cause inadvertent switching in logic circuits. To mitigate single-event effects RadPC uses an architectural approach in

which the detection and recovery of SEEs is abstracted from software developers. RadPC is developed in VHSIC Hardware Description Language (VHDL) and implemented as a logic design. RadPC expands upon the traditionally used Triple Modular Redundancy (TMR) to Quad Modular Redundancy (QMR) to withstand two SEEs simultaneously (Figure 1). RadPC is a QMR microcontroller implemented on an FPGA with a RISC-V instruction set [13], [14]. When a fault is detected in one of RadPC's CPUs its respective registers are reloaded with the correct values, using a majority voter. In the event of an unrecoverable fault using the voting approach, a full CPU can be partially reconfigured (PR'd) and registers to be reloaded to recover the system fully [15], [16]. These two recovery methods, complemented by memory scrubbers, provide resilience from SEEs [12], [13], [15], [16].

B. Cybershield

The architectural approach taken with RadPC to resist and recover from radiation-induced faults was extended into Cybershield, enhancing its resilience against malware attacks, particularly buffer overflow exploits. Cybershield employs instruction code obfuscation by assigning unique opcode offsets to each core and updating the software binaries accordingly, preventing uniform opcode execution and adding resistance to injection attacks [17]. This approach leverages the flexibility of FPGA-based implementations and RISC-V architecture to ensure that, even if a vulnerability is exploited, opcode discrepancies among cores trigger immediate detection and recovery mechanisms. The Cybershield microcontroller runs multiple obfuscated cores in parallel, with an anti-voter module monitoring instruction registers to detect any anomaly caused by malware injection [18]. Malware that is injected into CyberShield will be replicated across each core and upon execution each core will see the same Opcode. This by design is impossible and will be flagged as an anomaly. This architectural innovation not only increases security through hardware-level diversity but also provides an additional defense layer by making unauthorized code execution significantly more difficult. By integrating RadPC's proven reconfiguration capabilities with advanced obfuscation techniques, Cybershield offers a robust solution for securing embedded systems in critical applications.

In the original proof of concept of CyberShield, the obfuscated instruction decoders and associated software binaries were embedded in the VHDL description of the system before implementation. This posed a problem because the system could not be bootloaded like a traditional microcontroller. Instead, changing the software required a new bitstream to be generated. This paper proposes a TEE that provides secure boot with obfuscated opcodes for CyberShield, bootloading it like a traditional microcontroller. These architectural changes to the original version of CyberShield enhance both security and reliability by addressing the bootloading challenges of the original version.

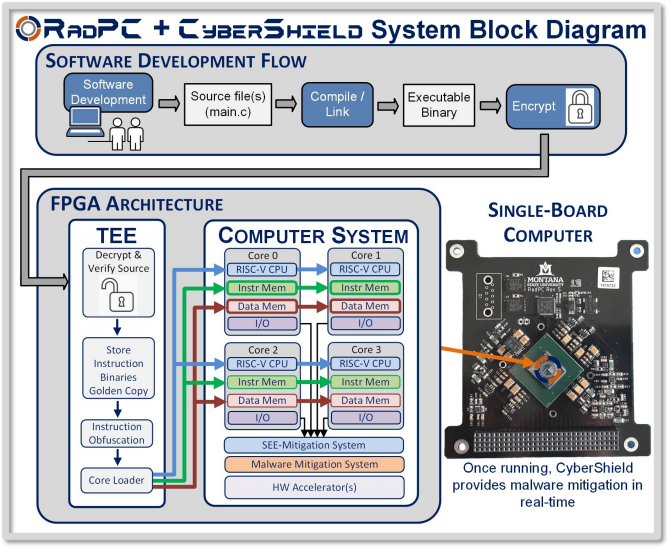

Fig. 2: System block diagram of RadPC + TEE i.e. Cybershield. Illustrates the software development flow, FPGA architecture, and the single-board computer.

IV. Design

The first modification to the CyberShield proof-of-concept was to modify the system so that it could have software binaries loaded into its instruction memory (e.g., bootloading) over a UART serial port. This allows the system to accept executables in real time. Based on this work, a TEE that bootloaded CyberShield was developed that could be implemented on the same FPGA. The TEE needed four key functionalities. First, to be able to accept an encrypted executable via a UART between CyberShield and an external computer. Second, to read and write the encrypted executable to the onboard NVM serial flash chip. Third, to be able to bootload CyberShield over an internal UART on the FPGA between the TEE and CyberShield's four cores. Last, the TEE needed to be able to obfuscate the software binaries before bootloading CyberShield.

The TEE was integrated into the CyberShield architecture as a separate softcore processor and deployed on a Nexys A7-100T development board. This board utilizes an Artix-7 Xilinx FPGA. The TEE and CyberShield are "wrapped" at the top level but are separate entities within the VHDL hierarchy. The architecture of the TEE is similar to the cores of CyberShield but with limited functionality and only necessary peripherals, i.e., SPI, UART, and GPIO. The other signals are tied to `0`, `(others => '0')`, or `open` for security. Both the TEE and CyberShield have been modified to include five UART peripherals. These UARTs are used by the TEE to bootload each individual CyberShield core instead of each core receiving the same executable.

The executable for the TEE is compiled using GCC on the developer's computer then the TEE is bootloaded over a UART. Once the TEE has been bootloaded it is ready to receive the encrypted executable for CyberShield. The

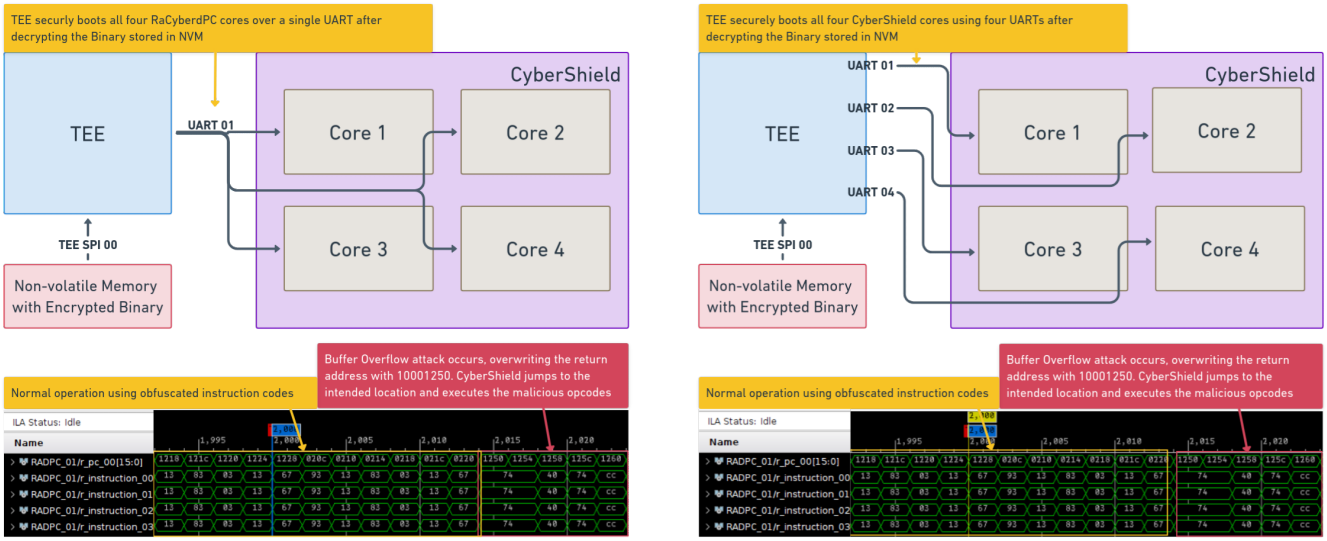

Fig. 3: Diagram of CyberShield and NVM coupled with their respective ILA outputs. The left block diagram shows CyberShield being bootloaded via one UART with no obfuscation. The respective ILA shows the four cores with the program counter running through a dummy function and the vulnerable overflow function and then falling victim to a buffer overflow attack. The right block diagram shows each individual CyberShield core being bootloaded via four UARTs with no obfuscation. The respective ILA shows the four cores with the program counter running through a dummy function and the vulnerable overflow function and then falling victim to a buffer overflow attack.

executable for CyberShield is first compiled using GCC and then encrypted using a one-time pad. The executable is then ready to be sent in one-byte blocks over UART to the TEE.

A. *Writing the executable to Non-volatile Memory*

In order for the TEE to accept the executable for CyberShield a circular buffer was implemented. The circular buffer was implemented due to memory constraints. The TEE and CyberShield share IMEM and DMEM which are both 12.288 kB. The test executable for CyberShield was 24.308 kB, making it too large for the TEE to accept in one block. Using a terminal, the developer can send blocks of up to 11520 bytes to the TEE to then be written over SPI to the NVM. The user specifies the size of the incoming block to the TEE, and then using a Python script sends the specified number of bytes over UART to the TEE at a Baudrate of 115200 with a one millisecond delay between bytes. The TEE stores the block of the executable in a byte array. Once the block of the executable has been accepted into the TEE it is ready to write to NVM.

The TEE communicates with the NVM via a Serial Peripheral Interface (SPI). The SPI peripheral on the TEE operates in 3-pin mode at 50 kHz, with the Chip Select (CS) line being manually toggled. Once a block of the executable is in the buffer, the developer can use the terminal to tell the TEE to write that block to the NVM. The TEE software keeps track of the number of bytes written along with the addresses to which that data was written. The data is written in 256-byte pages, and after each page, the TEE must toggle CS and reestablish communication with the NVM. Once the program is done writing to NVM, the circular buffer is reset and ready to receive another block of the executable. This process is repeated until the entire encrypted executable has been stored. Once this process is complete, the executable will remain in memory until it is cleared. The TEE is now ready to read from NVM and bootload CyberShield.

B. *Booting CyberShield with an Encrypted Executable*

1) Booting CyberShield Over One UART Without Offset (Figure 3: left): The first step in testing the system was to verify that the TEE could bootload the CyberShield QMR system without any instruction code obfuscation. To bootload CyberShield from NVM, the TEE requires two key components: the one-time pad decryption key and the instruction code offsets specific to CyberShield. Both of these have been manually programmed into the TEE software. The CyberShield cores' bitstreams are generated with their respective offsets when the executable is compiled.

To successfully bootload CyberShield the TEE must communicate using two different communication protocols: SPI and UART. When the executable was generated each 32-bit (four bytes) instruction was broken into four, single-byte pieces. The TEE reads the encrypted executable one byte at a time from the NVM and decrypts it in real-time. Initially, this test was performed using a single UART interface to transmit instructions to the CyberShield system without any instruction offsets.

2) Booting CyberShield Over Four UART Without Offset (Figure 3: right): After verifying that CyberShield could be bootloaded using a single UART, the next step was to test bootloading over four separate UART channels, one for each

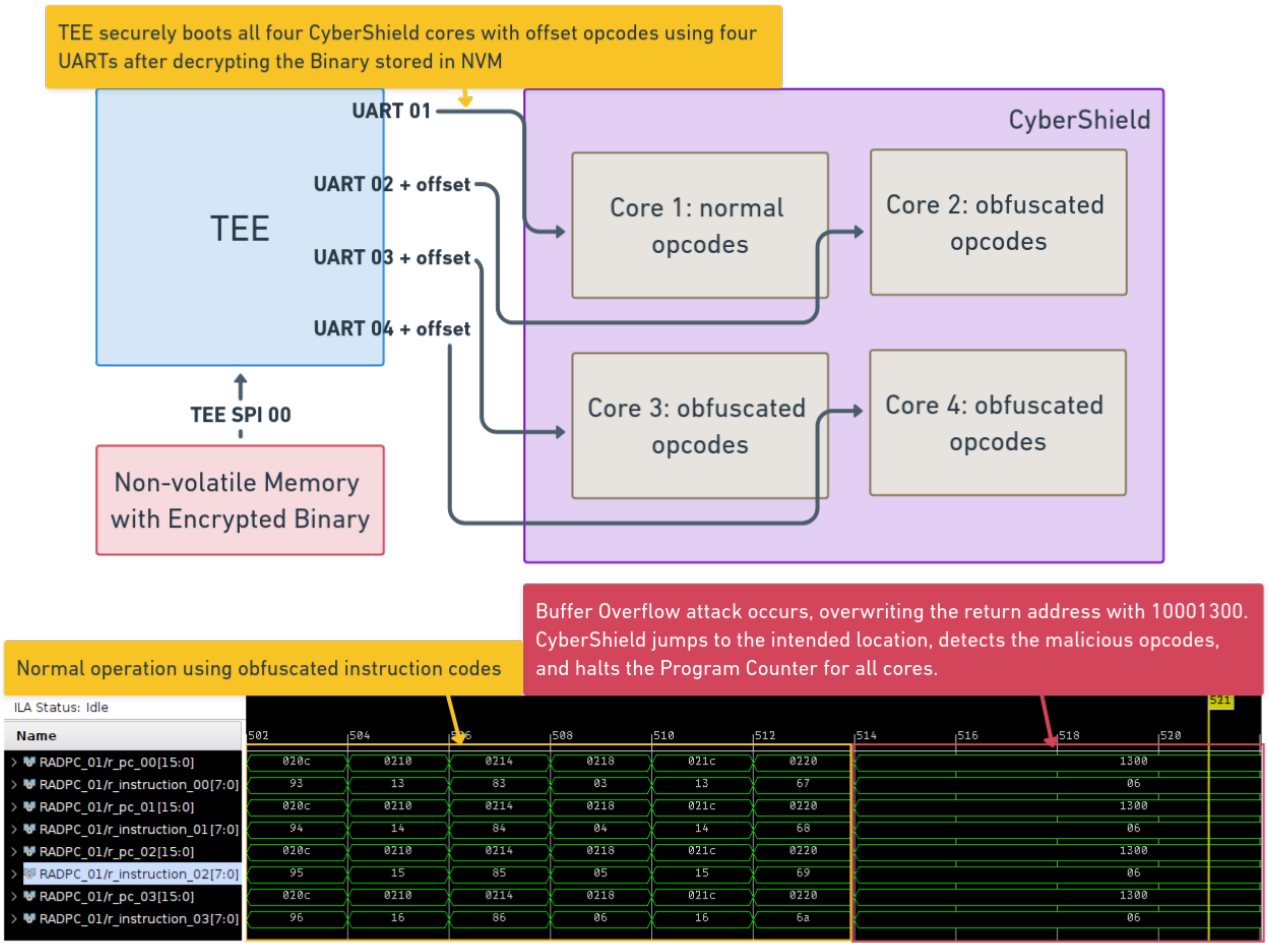

Fig. 4: Diagram of the CyberShield architecture with CyberShield and TEE integration, accompanied by ILA outputs. The diagram illustrates how the TEE securely bootloads each CyberShield core with its assigned opcode offset: Core 1 executes standard RISC-V opcodes, while Cores 2–4 operate with unique, offset opcodes. The ILA shows the cores running offset opcodes during normal operation. When a buffer overflow attack occurs, all four cores are forced to execute identical opcodes. The anti-voter in CyberShield detects that all four cores are running the same opcode and freezes the program counter to prevent the cores from executing the malicious code.

core. The process remained similar, but instead of using a single UART for all cores, the TEE distributed the executable to each core over their respective UART channel. At this stage, no offsets were applied to the instructions, meaning all four cores received identical instruction data. This setup more closely resembled the intended operational configuration of the system and allowed for independent verification of each core's ability to execute the received instructions. The successful execution of this setup demonstrated that the TEE could correctly handle multi-channel UART communication while maintaining synchronization.

3) Booting CyberShield Over Four UARTs (Figure 4): The final test introduced instruction code obfuscation to assess its impact on the bootloading process. Figure 4 presents the results of bootloading CyberShield with obfuscated instructions. In this stage, the TEE applied predefined offsets to the instruction sequences before transmission. Three of the four cores received obfuscated instruction data, while one core continued to run standard RISC-V 32I instructions. The obfuscation process altered only the last byte of each 32-bit instruction, ensuring that execution remained functionally equivalent but obscured from direct analysis.

Additionally, the TEE had to determine when to stop applying offsets to avoid corrupting non-executable sections of memory. The `ret` instruction sequence `(67 80 00 00)` served as a marker for the end of the executable section. Once the final occurrence of this sequence was detected, the TEE ceased applying offsets, leaving the following DMEM data unaffected.

Once all four cores were bootloaded with their respective

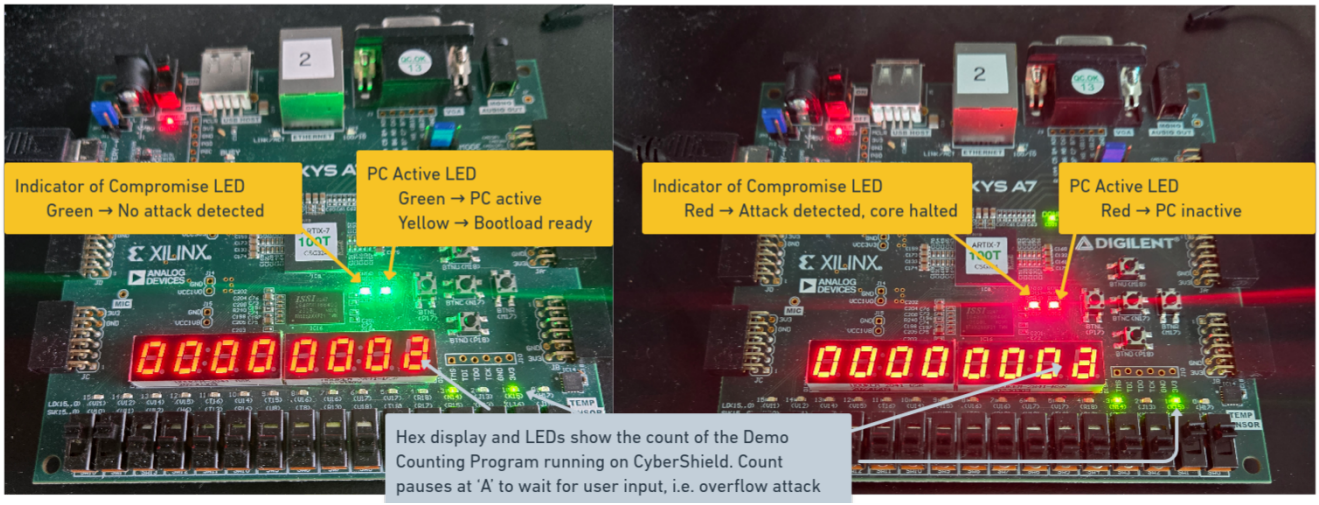

Fig. 5: Two Nexys A7 boards demonstrating CyberShield's runtime behavior. Left board: The green LED on the left indicates that CyberShield has booted successfully and no indicator of compromise (IoC) has been detected. The green LED on the right functions as a PC active flag, signifying that the cores are currently executing instructions normally. Right board: In contrast, the left red LED indicates that an IoC has been detected, signaling that the system has identified an attack. The right red LED, which normally serves as the PC active flag, now indicates that all cores have been halted to prevent further execution of potentially malicious instructions.

obfuscated instructions, CyberShield executed the program successfully, confirming that the obfuscation mechanism did not interfere with normal execution while increasing security.

C. Hardware Error Handing, e.g Anti-Voter

While the software for the TEE plays a crucial role in providing security and redundancy to CyberShield there are also critical hardware components. RadPC contains a voting system for each of its four cores in order to provide hardware redundancy in the case of a SEE. This voter compares all the instructions to ensure agreement and then passes the confirmed instructions. The anti-voter, inspired by RadPC's voting system, does the exact opposite. The anti-voter checks that no two instructions are the same as they are executed. If any of the cores share an instruction, this is an Indicator of Compromise (IoC), and an error flag is raised. When the error flag is raised, two actions occur: (1) the PC is halted to prevent CyberShield from executing any malicious instructions, and (2) both the PC active LED and the IoC LED turn red (Figures 4 and 5). In the future, this flag will be used by the TEE to reboot CyberShield after an attack.

V. Results

Cybershield was deployed on a Nexys A7 FPGA Development Board featuring the Artix-7 100T FPGA for testing. Cybershield was loaded onto the FPGA using Vivado Design Suite. Bootloading the TEE and loading the executable into the TEE for NVM were done over the serial terminal. A demo counting program was created to simulate CyberShield's software.

A. Buffer Overflow Attack

To test the resiliency of the obfuscated opcodes for CyberShield a buffer overflow attack was designed to overwrite the stack and have the Program Counter (PC) return to a location outside of Instruction Memory. The payload for the buffer overflow attack was delivered over UART 04 on CyberShield from a serial terminal. A switch was used to change the serial terminal windows connection between UART TEE 00 and CyberShield UART 04 so that CyberShield could be attacked.

The attack to be delivered over the serial terminal was on a modified version of the test counting program. This program contained a "Dummy" function that had a `printf` statement with unregulated input length. This function would continue to write to memory beyond the length of the buffer as long as data was being supplied. The disassembly for the test counting C program was used to determine where outside of Instruction Memory the program would return to when attacked. The stack and surrounding data were repeatedly overwritten with the address `10001250` which is a location in Data Memory (Figure 3 & 4.

The attack was tested on three different subsets of CyberShield. First, the attack was tested on CyberShield in which all four cores had been bootloaded with a single UART (Figure 3: left) and didn't have instruction obfuscation. Second, the attack was tested on CyberShield in which all four cores were bootloaded using separate UARTs without obfuscation (Figure 3: right). Last, the attack was tested on the full Cybershield system (Figure 4). When Cybershield is attacked the PC jumps to the address in Data Memory and attempts to execute the "instruction" at that location. As a result, the opcodes

of all four CyberShield cores become identical, indicating compromise.

This attack demonstrated how the obfuscation of the cores can defend against buffer overflows and other types of injection attacks. In the absence of a stack guard or address space layout randomization (ASLR), the redundancy of a quad-core system with obfuscated opcodes can defend against overflow and injection attacks. The ability to securely boot and reboot in the event of an attack ensures the continued operation of Cybershield even in the event of compromise.

VI. Conclusion

The principal results of this work demonstrate a proof-of-concept design for integrating a Trusted Execution Environment (TEE) within the RadPC architecture to create CyberShield. Once integrated, the TEE provided secure boot of obfuscated instructions to detect and defeat injected malware. This implementation also enhances the security of CyberShield by enabling secure boot using an encrypted executable. The TEE also provides a mechanism for software recovery in the event of an IoC. These enhancements collectively strengthen the resilience of CyberShield against a range of cyber threats, especially command injection and buffer overflow attacks.

A. Applications & Advantages

The addition of a TEE into CyberShield offers significant advantages over the baseline RadPC architecture. As an embedded system, RadPC lacks many security features that are standard in general-purpose computing platforms, such as Address Space Layout Randomization (ASLR) and StackGuard, which serve as defenses against injection and buffer overflow attacks [19]. By incorporating a TEE, CyberShield gains secure boot, in conjunction with opcode obfuscation, which provides real-time protection against such attacks [20]. This, in conjunction with, the capability to securely boot and reboot from NVM with an encrypted executable ensures robust software redundancy. Secure boot and encryption are particularly valuable for embedded systems that rely on persistent storage and may lack frequent software updates.

B. Limitations

Despite the security enhancements introduced by the TEE, certain limitations remain. This proof-of-concept implementation has not been extensively tested under adversarial conditions, leaving open questions about its resilience to advanced side-channel attacks and hardware-level exploits. The buffer overflow attack was detected, but further testing is needed to ensure it can detect other IoCs. While opcode obfuscation enhances security, it introduces compatibility challenges when integrating with existing software toolchains or debugging frameworks. Additionally, the integration of a TEE imposes additional computational overhead, which may affect performance, particularly in resource-constrained embedded environments. This system runs on a 100T FPGA, and smaller chips may not be conducive to Cybershield.

C. Future Work

Future research will focus on refining the security and performance of the CyberShield TEE implementation. Detecting other types of attacks outside of buffer overflows with ensure a more robust security framework. Moreover, adapting this TEE framework to other embedded architectures and security-critical applications, such as aerospace, medical devices, and industrial control systems, could broaden its impact. Finally, improving software recovery mechanisms through runtime integrity verification and anomaly detection techniques will further strengthen CyberShield's resilience against software compromises. Addressing these areas will ensure that CyberShield remains a secure and adaptable embedded computing platform capable of countering modern cyber threats.

VII. Acknowledgments

This research was supported in part by NASA under award number 80NSSC23CA147 and by Resilient Computing, LLC under subcontract number 4W9082. Any opinions contained herein are those of the authors and do not necessarily reflect those of NASA or Resilient Computing, LLC.

References

[1] C. Xu, S. Jiang, G. Luo, G. Sun, N. An, G. Huang, and X. Liu, "The case for fpga-based edge computing," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2610–2619, 2022.

[2] A. Ltd., "What is fpga?" [Online]. Available: https://www.arm.com/glossary/fpga

[3] J. Schneider and I. Smalley, "What is a field programmable gate array (fpga)?" May 2024. [Online]. Available: https://www.ibm.com/think/topics/field-programmable-gate-arrays

[4] S. Sunkavilli, Z. Zhang, and Q. Yu, "New security threats on fpgas: From fpga design tools perspective," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 278–283.

[5] J. Zhang and G. Qu, "Recent attacks and defenses on fpga-based systems," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 12, no. 3, pp. 1–24, 2019.

[6] P. Jauernig, A.-R. Sadeghi, and E. Stapf, "Trusted execution environments: Properties, applications, and challenges," *IEEE Security Privacy*, vol. 18, no. 2, pp. 56–60, 2020.

[7] G. Perkins, B. Macht, L. Ritzdorf, T. R. Crane, B. LaMeres, C. Izurieta, and A. M. Reinhold, "Sok: Trusted execution in soc-fpgas," 2025. [Online]. Available: https://arxiv.org/abs/2503.16612

[8] T. L. Loo, M. K. Ishak, and K. Ammar, "Design and implementation of secure boot architecture on risc-v using fpga," *Microprocessors and Microsystems*, vol. 101, p. 104889, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933123001321

[9] A. Dave, N. Banerjee, and C. Patel, "Care: Lightweight attack resilient secure boot architecture with onboard recovery for risc-v based soc," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 516–521.

[10] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: an open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387532

[11] F.-J. Streit, F. Fritz, A. Becher, S. Wildermann, S. Werner, M. Schmidt-Korth, M. Pschyklenk, and J. Teich, "Secure boot from non-volatile memory for programmable soc architectures," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2020, pp. 102–110.

[12] H. A. Austin, "Fault injection system for fpga-based space computers," Ph.D. dissertation, Montana State University-Bozeman, College of Engineering, 2023.

[13] C. M. Major, A. Bachman, C. Barney, S. Tamke, and B. J. LaMeres, "Radpc: A novel single-event upset mitigation strategy for field programmable gate array–based space computing," *Journal of Aerospace Information Systems*, vol. 18, no. 5, pp. 280–288, 2021. [Online]. Available: https://doi.org/10.2514/1.I010859

[14] J. Williams, C. Barney, Z. Becker, J. Davis, C. Major, B. J. LaMeres, and B. Whitaker, "Radpc@scale: A novel approach to radpc single event upset mitigation strategy," *2022 IEEE Aerospace Conference*, 2022.

[15] B. J. LaMeres and C. Gauer, "Dynamic reconfigurable computing architecture for aerospace applications," *2009 IEEE Aerospace Conference*, 2009.

[16] C. Gauer, B. J. LaMeres, and D. Racek, "Spatial avoidance of hardware faults using fpga partial reconfiguration of tile-based soft processors," *2010 IEEE Aerospace Conference*, 2010.

[17] L. L. Ritzdorf, C. Barney, C. M. Major, T. R. Crane, H. Austin, B. Macht, C. Izurieta, and B. J. LaMeres, "Evaluating the effectiveness of obfuscated instruction codes for malware resistance," in *2023 Intermountain Engineering, Technology and Computing (IETC)*, 2023, pp. 67–72.

[18] T. T. Running Crane, "Using instruction code obfuscation to defeat malware attacks," Ph.D. dissertation, Montana State University-Bozeman, College of Engineering, 2023.

[19] M. A. Butt, Z. Ajmal, Z. I. Khan, M. Idrees, and Y. Javed, "An in-depth survey of bypassing buffer overflow mitigation techniques," *Applied Sciences*, vol. 12, no. 13, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/13/6702

[20] Z. Shao, Q. Zhuge, Y. He, and E.-M. Sha, "Defending embedded systems against buffer overflow via hardware/software," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, 2003, pp. 352–361.