VERIFICATION TOOL FOR SECURING RISC-V FPGA-BASED PROCESS CONTROL SYSTEMS

by

Christine Lois Johnson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2025

ACKNOWLEDGEMENTS

I would like to express deep gratitude to my advisor and the chair of my graduate committee, Dr. Clemente Izurieta, for his support and direction throughout the development of this thesis. I also wish to extend my gratitude to Dr. Brock LaMeres for his involvement in my graduate committee and guidance in developing the physical control system for my thesis. Lastly, I would like to thank Dr. Bradley M. Whitaker for his involvement in my graduate committee.

Funding for this research is made possible by support from Resilient Computing, founded by Dr. LaMeres, and the Software Engineering and Cybersecurity Laboratory (SECL) at Montana State University, co-directed by Dr. Izurieta and Dr. Ann Marie Reinhold. 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. My final thanks goes to the members of the SECL and Resilient labs, especially Garrett Perkins and Tristan Running Crane, for their aid in the technical development of this thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

NOMENCLATURE

AitM	Attacker-in-the-Middle
ADC	Analog-to-Digital Converter
BIN	Binary
BJT	Bipolar Junction Transistor
C	Charging Rate
CIA	Confidentiality, Integrity, and Availability
CWE	Common Weakness Enumeration
DoS	Denial-of-Service
FPGA	Field Programmable Gate Array
FTDI	Future Technology Devices International Limited
GCC	GNU Compiler Collection
HDL	Hardware Description Language
HMI	Human Machine Interface
ICS	Industrial Control System
IoT	Internet-of-Things
ISERN	International Software Engineering Research Network
I/O	Input and Output
IT	Information Technology
mAh	Milliampere-hour
NiHM	Nickel-Metal Hydride
NPN	Negative-Positive-Negative
OT	Operational Technology
PAC	Programmable Automation Controller
PCS	Process Control System
PLC	Programmable Logic Controller
PMOD	Peripheral Module
RISC-V	Reduced Instruction Set Computer Five
RTU	Remote Terminal Unit
SECL	Software Engineering and Cybersecurity Lab
TAP	Test Access Point
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
V	Volts
YARA	Yet Another Recursive Acronym

ABSTRACT

Industrial Control Systems (ICSs) are a crucial critical infrastructure component and a popular cyberattack target. While most research in this area focuses on the defense of large-scale networks of ICSs, it is critical to also expand research on the small-scale networks of Process Control Systems (PCSs), which attackers may target to remain undetectable from the security of the more extensive network. One potential protection method for PCS networks is the use of verification tools; however, existing research on verification tools focuses solely on detecting attacks without mitigation. This research describes two instantiations of a verification tool for a PCS controlled by a RISC-V computer implemented on an FPGA. An experimental testbed was developed to test the tools, consisting of (1) a circuit to charge a battery, (2) an FPGA controller that controls charge/discharge based on user input, (3) an engineering station that provides control data and updates firmware to the FPGA; and (4) a verification tool that verifies input forwarded by a passive serial tap, connected through the FPGA's hardware. The first version of the verification tool provides passive detection, whereby it detects and informs the engineering station of an attack. The second version provides detection and mitigation against unauthorized command messages and malicious software downloads. The experimental data yielded promising results, with the tool successfully providing mitigation and detection against attacks on the serial communication channel between the engineering station and FPGA. This approach applies to standard ICS computer devices, such as programmable logic controllers (PLCs).

1

INTRODUCTION

Industrial Control Systems (ICSs) are an integral part of a nation's infrastructure and are expected to operate continuously and without error. Many industrial systems were originally established as independent and standalone Operational Technology (OT) systems. Still, the rise of modern-day Information Technology (IT) has allowed them to become increasingly interconnected with each other through local networks and the Internet. Although this interconnectedness allows for new advances in ease of use and access to a system, it also increases the surface area of attacks. Most of the research on security around ICSs focuses on their larger network systems to prevent attackers from gaining large-scale access to the control system. However, this can lead to neglect of well-rounded security for smaller system components, as it is assumed that attacks will be stopped well before they reach that level.

One of these smaller components, and an essential subcomponent of ICSs, is the Process Control System (PCS). PCSs control simple, singular processes within a larger ICS, such as raising or lowering the level of a water tank within a sewage plant. They are typically controlled with Programmable Logic Controllers (PLCs), which use the innately insecure Modbus communication protocol [18]. Attacks against PCSs are well documented within the field, with the Stuxnet attack against an Iranian nuclear facility in 2010 being one such notable example [5]. This attack caused physical damage to the facility through a vulnerability found within a Siemens PLC-controlled PCS. In particular, many attacks take advantage of PCSs due to their lack of comprehensive implementation of security protocols. An area that is lacking is the Internet-facing security on PLCs and other components, which is sometimes ignored or disabled because it interferes with the operations of the components. A

constant issue faced when adding more security to control systems occurs when incorporating security measures, which has a negative impact on the availability of the systems.

One interesting method to ensure the integrity of measurement data and control logic delivered along communication channels to PLCs without impacting the operation of the system is verification tools. This is due to the passive nature of verification tools, which monitor the control data that is provided to a PLC or similar controller without preventing or delaying its arrival to the controller. Research on verification tools currently explores the ability to detect attacks that have occurred, but research on their use to prevent attacks has not received the same attention.

The goal of this research is to create a verification tool that provides mitigation of these attacks in addition to detection without negatively impacting the control system's availability. This thesis describes two instantiations of a verification tool for a PCS controlled by a RISC-V computer implemented on a Field Programmable Gate Array (FPGA). Separately, these instantiations provide protection against attacks: passive detection, where the tool detects and informs the engineering station of an attack, and detection with mitigation, where the attacks never reach the controller and the engineering station is informed. These tools operate in the presence of unauthorized command message attacks in which an attacker aims to modify the control logic of the controller within the system. Though this tool was developed for a RISC-V FPGA, the approach applies to other standard ICS computer devices, such as programmable logic controllers (PLCs), as it uses the serial communication protocol commonly used by these devices.

This thesis is organized as follows. Section 2 covers background and related literature. Section 3 describes the technical descriptions of the designs for each component in the experimental testbed. The results of the implemented design are provided in Section 4. Conclusions of the thesis are given in Section 5, and possibilities for future work are covered in Section 6.

3

BACKGROUND AND MOTIVATION

An ICS comprises complex hardware and software components designed to automate processes within physical industrial systems. Within the cybersecurity landscape, ICSs have received significant attention, primarily because of their role in national infrastructures and the surge in highly visible and harmful cyber-assaults targeting these systems. In particular, Stuxnet stands out as one of the most documented examples of such attacks [5].

The Stuxnet attack occurred in 2010 and targeted an Iranian nuclear facility, causing physical damage to the facility by taking advantage of a vulnerability within the facility's Siemens PLC controlled PCSs to spread malicious control logic. PLCs are one of the common components used in ICS PCSs. These peripheral devices used in PCSs are typically equipped with internet-facing components but are not always secure by default. This allows attackers to have direct access to logic controls with the added advantage of avoiding the security of the more extensive ICS network. These attacks have become increasingly commonplace as traditionally separate OTs and ITs converge, exposing legacy systems to new threat vectors. We discuss basic background information on PCSs, including common weaknesses, attack techniques, and defense techniques, as well as developing a fundamental understanding of current research and development of verification tools for ICS controllers.

Process Control Systems

In ICSs, a PCS is an individual control system within the larger ICS that controls a simple process. For example, in a water treatment facility, a PCS would control a simple process within the system, such as raising or lowering the level of a water tank. Currently, a large majority of research for PCSs focuses on PLCs [3], which are traditionally programmed using ladder logic.

Other implementations of PCSs typically rely on components such as Remote Terminal

Units (RTUs) and Programmable Automation Controllers (PACs) to serve as their controllers [33, 39]. RTUs are microprocessor-controlled electrical devices that can monitor and control connected devices by communicating telemetry data to a control system and changing the states of devices based on received control messages. In general, they are similar in function to PLCs but are typically more durable and thus preferred for use in remote locations and environments with extreme temperatures [7, 20]. PLCs and RTUs usually make use of the Modbus communication protocol, which lacks built-in strong authentication and encryption mechanisms [18]. Lastly, PACs are also similar in function to PLCs, but are more powerful and intricate, with processors, memory, and software similar to a PC. PLCs are typically programmed in C or C++ rather than ladder logic [10].

For simplicity of programming, as the internal logic of many PLCs is not easily accessible or available from vendors, this research focuses on a control system where the controller is implemented with an FPGA. FPGAs are built around a matrix of configurable logic that allows them to be programmed and reprogrammed as needed. Thus, they can be specialized to fit the specific needs of a control system. Although FPGAs are used less commonly as controllers in ICSs, as PLCs tend to be cheaper and easier to use, FPGAs can be programmed to perform the same functionality within an ICS as a PLC [30, 31]. As noted in this review [29], FPGAs have been used in many ICSs and continue to be considered as controllers by an increasing number of designers in various fields. In particular, the specific RISC-V-based FPGA computer used in this research would be more useful as an ICS controller utilized in the presence of radiation, as its functionality has fail-safes against radiation [12].

Weaknesses

When discussing PCS weaknesses, we refer to the nomenclature used in the MITRE CWE, Common Weakness Enumeration, a collection of software and hardware weaknesses that refers to common weaknesses that may lead to vulnerability incidents [15]. In recent

years, PCSs within ICSs have been implemented with greater interconnectivity, which improves their ease of use and remote use. In conjunction with these improvements, though, comes more exploitable vulnerabilities on the PCSs, which had previously faced mainly internal attack vectors owing to protective air gaps in the system.

To incorporate internet-facing PLCs into PCSs without negative effects on the control system's performance, many systems chose to ignore or disable their security features, leaving them further open to attacks [25]. This relates to CWE-655: Insufficient Psychological Acceptability, where a product has a protection mechanism that is too difficult or inconvenient to use, encouraging non-malicious users to disable or bypass the mechanism, whether by accident or on purpose. With this lack of adequately implemented anomaly detection for PLCs to identify irregular control logic and measurements, methods of anomaly detection for PCSs have been a well-studied topic [9].

Another common weakness experienced in PCSs is based on a lack of proper authentication and encryption that is innate to its communication protocols. Many systems use the Modbus communication protocol, which does not offer built-in encryption or strong authentication mechanisms [5, 44]. This relates to weakness CWE-311: Missing Encryption of Sensitive Data and allows attackers easy access to gather information on a system's measurements and commands, allowing for easier fabrication of attacks.

PCSs also experience weaknesses due to human factors, such as malicious personnel and inexperienced users. Before the 2000s, when the air gap between OT and IT was still largely present, these internal human factors were the most prevalent weaknesses. However, as explored above, with increasing integration between OT and IT, external attacks have come to far outnumber internal attacks [6].

Attack Techniques

There are a plethora of attack techniques available against PCSs. Most attacks on PCSs have the end goal of affecting the availability portion of the CIA triad, which consists of Confidentiality, Integrity, and Availability [3]. The concept of ICS availability is best described with a discrete math function that answers the question: *What is the probability that a system is up and running at some point in time?* Availability for ICSs is critical, as systems need to run uninterrupted to maintain proper functionality. The availability of an entire ICS can even be affected through just one of its PCSs, as many ICSs depend upon the constant and correct operation of individual processes. The previously cited Stuxnet example is one such attack in which the availability of an ICS was targeted through a PCS, causing a delay in power generation at the Iranian nuclear power plant.

Although Stuxnet is the most infamous, the first recorded case of a deliberate cyberattack against critical infrastructure, targeting its control system, was in 2000. A dissatisfied contractor at a Queensland sewer injected false commands and data into a wastewater station, causing one million liters of wastewater to be expelled into nearby waterways [35]. A less impactful but more recent example is from 2023. In this case, cyber actors using the persona "CyberAv3ngers" began actively targeting and compromising Israeli-made Unitronics Vision Series PLCs and engineering stations, which are commonly used in water and wastewater systems and other important industries [22]. These actors left a defacement image, which didn't cause physical harm to the PCSs. However, with this type of access, more profound cyber-physical effects on the PCS could be performed if desired. It's believed that the actors took advantage of poor password security and exposure to the internet on the devices, and connected to them remotely using Unitronic's engineering workstation software to download the malicious project to the PLC. One American critical infrastructure system included in these attacks was the Municipal Water Authority of

Aliquippa in Pennsylvania, which provides water treatment to facilities in multiple townships [43]. The ICS's programmable logic controllers can be used to turn off pumps at a pump station to fill tanks and reservoirs, flow the pace of chemicals, and gather monthly compliance data. Because of this, an attack by these actors with the intent of physical damage could negatively impact the water supply of the city.

A frequent form of attack launched against PCSs is Adversary-in-the-Middle (AiTM), also known as Man-in-the-Middle, attacks, especially those with internet-facing PLCs [45]. In AiTM attacks, attackers with privileged network access can read or spoof network traffic. A large portion of attacks on PCSs consist of false data injection and control logic injection attacks, which are typically a form of AiTM attacks. In the ICS ATT&CK Matrix, these correspond to the Spoof Reporting Message and Unauthorized Command Message techniques, which fall within the Impair Process Control tactic (affecting availability) [16]. A portion of the matrix with highlighted attacks is given in Figure 1. With the spoof reporting message technique, an attacker seeks to cause harm by altering reported measurement data used as feedback to control a system, causing improper commands to be performed in response. A specialized form of spoof reporting attacks is replay attacks, in which an attacker who has obtained legitimate past measurements from a sensor sends that data back into the sensor in place of actual current measurements [42]. With unauthorized command message attacks, an attacker sends malicious commands to cause assets within the system to perform actions that are not aligned with the current state of the system or completely beyond its normal bounds [2].

Lastly, we'll consider the Denial of Service (DoS) attack technique. These attacks disrupt a system's ability to respond appropriately to actions within it by sending a high volume of traffic that the system is incapable of handling. Some notable attacks against PCSs have occurred. One such incident was the 2003 attack on the Divs-Besse nuclear power plant in Oak Harbor, Ohio [18]. The plant was infected with the 'Slammer' worm,

Evasion	Discovery	Lateral Movement	Collection	Command and Control	Inhibit Response Function	Impair Process Control	Impact
Change Operating Mode	Network Connection Enumeration	Default Credentials	Adversay-in-the-middle	Commonly Used Port	Activate Firmware Update Mode	Brute Force I/O	Damage to Property
Exploitation for Evasion	Network Sniffing	Exploitation of Remote Services	Automated Collection	Connection Proxy	Alarm Suppression	Modify Parameter	Denial of Control
Indicator Removal on Host	Remote System Discovery	Hardcoded Credentials	Data from Information Repositories	Standard Application Layer Protocol	Block Command Message	Module Firmware	Denial of View
Masquerading	Remote System Information Discovery	Lateral Tool Transfer	Data from Local System		Block Reporting Message	Spoof Reporting Message	Loss of Availability
Rootkit	Wireless Sniffing	Program Download	Detect Operating Mode		Block Serial COM	Unauthorized Command Message	Loss of Control
Spoof Reporting Message		Remote Services	I/O Image		Change Credential		Loss of Productivity and Revenue

Figure 1: A portion of the MITRE ICS ATT&CK Matrix relevant to PCS attacks. The matrix contains columns of attack techniques labeled by their intended tactic. The highlighted techniques include Adversary-in-the-Middle and two techniques it furthers that seek to impair process control and may cause physical impact to an ICS through its PCS.

which resulted in a traffic overload on the site network, leaving the safety monitoring system of the plant inaccessible for about five hours.

Defense Techniques

Current defense techniques typically focus on either detection or mitigation of intrusions, also referred to as prevention, with the majority of techniques focusing on the former [3]. Detection techniques focus on recognizing or identifying attack tactics. Among the standard intrusion detection methods are honeypots, verification tools, and dynamic watermarking. Honeypots are intended to lure attackers in and allow an attack to occur in a safe setting so that information can be safely collected on the attack [28]. Another method being explored is including more thorough network traffic inspection, such as overshadowing that can monitor for control logic code in ICS network traffic [46]. Machine learning has also been explored as a method for identifying attacks that involve false data measurements or

control logic [34].

In addition to detection techniques, mitigation techniques serve as another form of defense in PCSs. These techniques focus on preventing or reducing harm from an attack tactic. One such mitigation technique is encryption for ICS networks. This technique typically addresses the problem of lacking encryption protocols within the system's network protocols, which often send plain text, by protecting content with encryption algorithms before it is sent. However, this technique can have high computational overheads [3]. Verification tools work to confirm the veracity of the firmware or software within the PCS, detecting malicious actions if any of these facets have been altered, typically falling within the Validate Program Inputs technique [27]. Dynamic watermarking attempts to determine the veracity of a signal within the PCS [24]. Within the system, the data is altered with a noise signal, which is expected within the system. The system will then alert to an attack if proper alteration to the signal is not detected.

Many PCS components are proprietary, and therefore, it can be challenging to develop generalized defense techniques that adapt quickly to separate types. In particular, this often prevents proper understanding of the operational logic and internal mechanisms of these embedded systems when developing defense techniques [41]. Thus, an area that needs to be further developed for PCS and its components is open-source, flexible defense techniques.

Serial Communication

The majority of PCS components use the Modbus protocol to communicate, and Modbus is used primarily over serial lines. Serial communication sends data over a line as a series of bits [26]. Serial data standards such as the 1962 Recommended Standard 232 (RS-232) standard and the 1996 Universal Serial Bus (USB) standard specify how serial data is transmitted over serial communication channels. RS-232 is no longer commonly used with most general computers, replaced by USB, but is still widely used in industrial monitoring

and embedded systems such as PLCs [17].

A network Test Access Point (TAP) is a device that connects directly to cabling infrastructure to split or copy packets for use in things such as analysis and security [19]. In particular, this project uses a network component that behaves like a passive serial tap. A serial tap is a network tap specialized for serial communication. Any passive tap is invisible to the rest of the network it is connected to, so it may monitor traffic undetected. Passive serial taps are fairly standard devices within PCSs, as serial communication is shared among its components. Typically, these taps exist as separate devices that can be physically wired and/or connected to the communication channels of the PCS.

RISC-V Architecture

RISC-V is a relatively new open standard instruction set architecture, introduced in 2014, which focuses on reduced instruction set computer concepts. A reduced instruction set computer is a computer architecture design that aims to simplify instruction sets while improving execution. Other features of RISC architectures typically include single-cycle execution, load-store architecture, many general-purpose registers, and pipelining, all helping towards the goal of efficient execution. RISC-V, in particular, has 32, 64, or 128-bit address spaces, and the integer core is extended with floating point, atomics, and vector processing. It is also designed to be modular and customizable, so it can be expanded upon with new instructions for networking, I/O, and data processing, among others [38].

RISC-V was a compelling choice for this research, as it is intended to be usable for most practical computer use cases, and it is not over-specialized towards any specific use case. RISC-V can be helpful when there is a need to reduce power consumption, code size, and memory use. The reduced code size of RISC-V, especially when using compressed instructions, can reduce the binary size of code for use on small computers. Thus, it is a reasonable consideration for use on deeply embedded systems like those we frequently

encounter in ICS. RISC-V heavily features modularity, beginning as a limited base instruction set that can be made more complex with optional extensions and/or the addition of custom instructions.

In this research, we selected the RadPC edge computer, developed by Resilient Computing, as our controller [12]. RadPC is implemented onto a Xilinx Nexys Artix-7 FPGA and its processor architecture is RISC-V RV32I ISA, which supports basic integer operations with 32-bit registers. For this work, it provided an I/O rich platform that looked like a single RISC-V CPU to the developer. The firmware of RadPC for this project is considered a black box. RadPC may also be configured to provide obfuscation as a form of protection against local code tampering. Additionally, it has fault-mitigation procedures that help it to continue operating in extreme environments.

Verification Tools

When concerning FPGAs, verification is typically used as part of the software development process to verify that an FPGA's logic code performs as expected and is a part of general FPGA Design Methodology [29]. However, this paper explores a different style of verification tools that may be used with controllers such as FPGAs and PLCs, a style that seeks to verify the integrity of remote code, such as logic code and firmware, sent to an FPGA or controller over an untrusted or unencrypted communication channel. One paper uses time for its verification function, which is built such that any modification of the program increases its runtime, alerting the tool of possible tampering [4].

Two papers heavily inspired the direction of research for this thesis. In PCS, remote codes are vulnerable to alteration by attackers along the routes of communication. One such paper that uses a verification tool to confirm the integrity of remote-code is [4]. This paper uses checksums to provide fingerprints for the remote codes. Another uses a verification tool to confirm just the firmware uploaded to PLCs over RS232 serial communication in

[27]. This paper uses SHA-256 hashing to provide fingerprints for the code. SHA-256 is a cryptographic hash algorithm that produces an almost unique 256-bit signature for a text or data file that can then be used to ensure file integrity [37]. Some newer papers explore using side-channel power analysis to inform their verification tools [21]. The development of a verification tool for remote code to the board code provides protection against remote tampering of code as well.

Rechargeable Batteries

Batteries can provide electrical energy to devices by converting stored chemical energy to electrical energy. Nickel-Metal Hydride (NiMH) batteries are a common type of rechargeable standard-size battery. Rechargeable batteries, once depleted, can convert supplied electrical energy back to stored chemical energy within themselves [32]. A battery discharges when it supplies more current across a load than it receives, and it charges when it receives more current than it supplies.

There are a few terms that will be defined to better describe working systems involving batteries. A battery's nominal voltage is the average voltage it may output when fully charged. A battery's capacity is the amount of electrical energy it can store and deliver by providing a certain discharge current from a fully charged state to a depleted state for a period of time. It is given in amp hours (Ah). The state of charge (SoC) of a battery expresses its remaining capacity as a percentage of its maximum capacity. The charge rate (C rate) of a battery describes the rate at which the battery discharges or charges in proportion to its maximum capacity, with 1C being a rate that will fully discharge the battery in one hour [40].

Montana State University Contributions

This research further expands upon the work of Montana State University's Software Engineering and Cybersecurity Laboratory (SECL)[1] and Resilient Computing[2]. SECL is an interdisciplinary team of computer and data scientists and software engineers researching problems using convergent scientific and engineering approaches in an Applied Research Lab for classified research. One focus of the lab is software quality assurance approaches to cybersecurity regarding both software and hardware prior to its deployment in critical infrastructure. Another has been the development of resilient computers with redundant hardware that monitor malicious activity, which continue to be developed in tandem with Resilient Computing.

Resilient Computing is a Montana State University spin-off company that commercializes edge computing technologies used in space and critical infrastructure. Namely, Resilient Computing has commercialized SECL's RadPC computer[12], a computer developed to reliably operate in the presence of space radiation, and Cybershield, which allows the computer to perform malware detection by using obfuscated instruction codes in the functionally equivalent processors [36]. Contributions from the SECL lab used in the development of this research include the previously developed RadPC core for use on Xilinx FPGAs and the related workflow toolchain for this core [13].

Motivation & Goal

PCSs are a popular target of attacks today, often through their PLCs. Many PLCs still lack significant protections, such as encryption of communication and anomaly detection of received communication. Even when some forms of protection are available, they are

[1] https://www.montana.edu/cyber/
[2] https://resilient-computing.com/

sometimes unused as they may negatively affect the control system's performance. This thesis was motivated by the need for more convenient prevention options against these attacks, as the availability of PCSs and ICSs is imperative to our nation's critical infrastructure.

One potential form of anomaly detection that may be applied to PLCs without negatively affecting their performance is through the use of verification tools. These tools have only provided detection of attacks in past research. The goal of this research is to expand the capability of this tool to provide mitigation as well as detection while having no impact on the performance of its PCS. This goal is worthwhile because, in PCSs, attacks must be prevented to ensure the PCS performs appropriately, but at the same time, the prevention methods cannot negatively impact the availability of the system. The tool presented in this thesis addresses both issues.

SYSTEM DESIGN

Overview

In this section, we define the PCS and the verification tool applied in this research. As the verification tools explored in this study pertain to PCSs, it was preferable to develop the verification tool in the scope of a simple PCS. Thus, a simple PCS was created in which the system controls the charge and discharge of a 9 V NiHM rechargeable battery, and the development of the verification tool was applied to this experimental testbed. Two different versions of the verification tool were designed, with the first providing passive detection and the second providing detection with mitigation. As some of the system's components vary for each verification tool version, a base description will first be given for each of the components, and then further description will be provided if there are differences or additional details pertaining to the separate versions of the tool.

System Components

We will give a quick overview of the hardware and software components that make up the system. The battery within the plant system is a NiMH 9 V rechargeable battery with a nominal voltage of 8.4 V, of which the specification sheet has been included in the footnote. The voltage sensor is provided by the FPGA analog input, which uses control logic provided by an engineer to turn the system's charger on or off. The battery charger is implemented as a variable voltage source, which is set to 12 V. The capacity of the 9V battery is 200 mAh, so the system was designed to not exceed a 200 mAh charging current or 1C charge rate. An output from the RadPC FPGA will allow the battery's charge flow to be turned on and off, depending on the battery's charge. A seven-segment display on RADPC indicates the current voltage of the battery, maximum and minimum limits for voltage charging, and the

current state of the charge signal.

Software components utilized in this project include a prebuilt RISC-V GCC Toolchain for Linux[5], a compiler that supports an RISC-V RV32I core, within a Windows Subsystem for Linux environment. This is used as part of a software toolchain workflow for the RadPC Reach RV32I softcore processor. Currently, we use a RadPC core designed by Resilient Computing, implemented onto a Xilinx Nexys Artix-7 FPGA. The processor architecture is RISC-V RV32I ISA, which supports basic integer operations with 32-bit registers.

In an earlier iteration of this project, a single Nexys A7 FPGA that hosts a RadPC core was used as a controller for the PCS, and another without RadPC was used as a peripheral voltage sensor. This was done to address the issue of RadPC not yet including an analog-to-digital converter (ADC) component. In the final stages of the project, ADC capability was developed for the RadPC, so the voltage sensor and the controller were both instantiated on the single Nexys A7 FPGA hosting RadPC instead. Additionally, an FTDI Chip C232HD-DDHSP-0 USB to UART Serial cable was used to provide an additional source of UART communication that will be explored more in the FPGA Logic section. Diagrams showing the connections of all system components for the initial and finalized designs are included in Figures 2 and 3. They show connections where PCS components are connected with wire, indicated with dotted ends, as well as the cable connections between the USB ports of the PCS components. Individual components of the finalized design will be covered more thoroughly in later design subsections.

Control System Design

The control system is represented as a negative feedback loop within the control block diagrams pictured in Figures 4 and 5. In Figure 4, we show a traditional negative feedback loop, which describes the PCS without external ICS parts, verification, and attack components. Figure 5 shows a modified version of the control loop, which shows how the

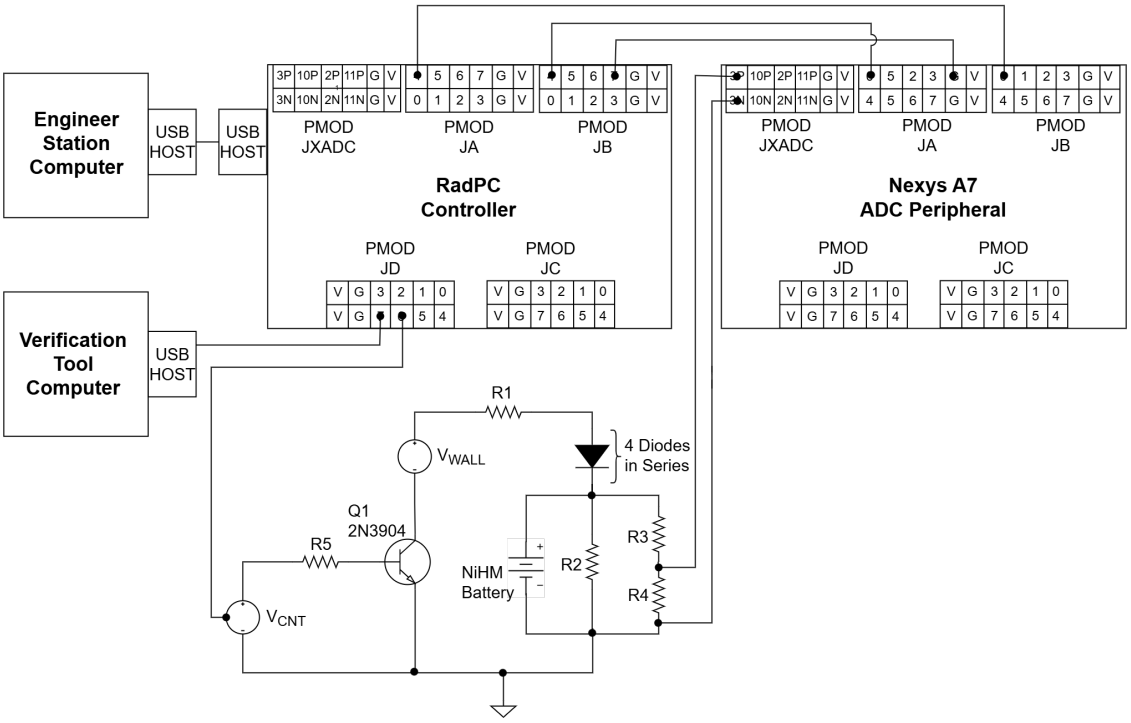

Figure 2: A schematic diagram of the initial process control system (PCS) design. The PCS was originally designed with two FPGAs, one with a RadPC core for controlling the PCS and one with the original Nexys A7 core as a peripheral for monitoring voltage. The GPIO of the boards consists of the labeled PMOD rectangles, with the individual pins labeled in the squares. Dots on the diagram represent where multiple components are joined. USB connections for communication are designated with a single line.

external components communicate with the control system. Components within the main control loop are colored black, whereas external components that communicate with the control system are depicted in color. The legitimate external sources of communication in this diagram are represented by green arrows, whereas adversary sources are represented by red arrows.

We first describe the components in the diagram that exist within the traditional control loop, which features five components. Components are represented within the control loop as rectangles. The Engineering Station block allows human users to send and receive data from the FPGA controller through a USB to UART bridge. The FPGA Controller block component outputs an on/off signal for the battery charger based on feedback logic from the

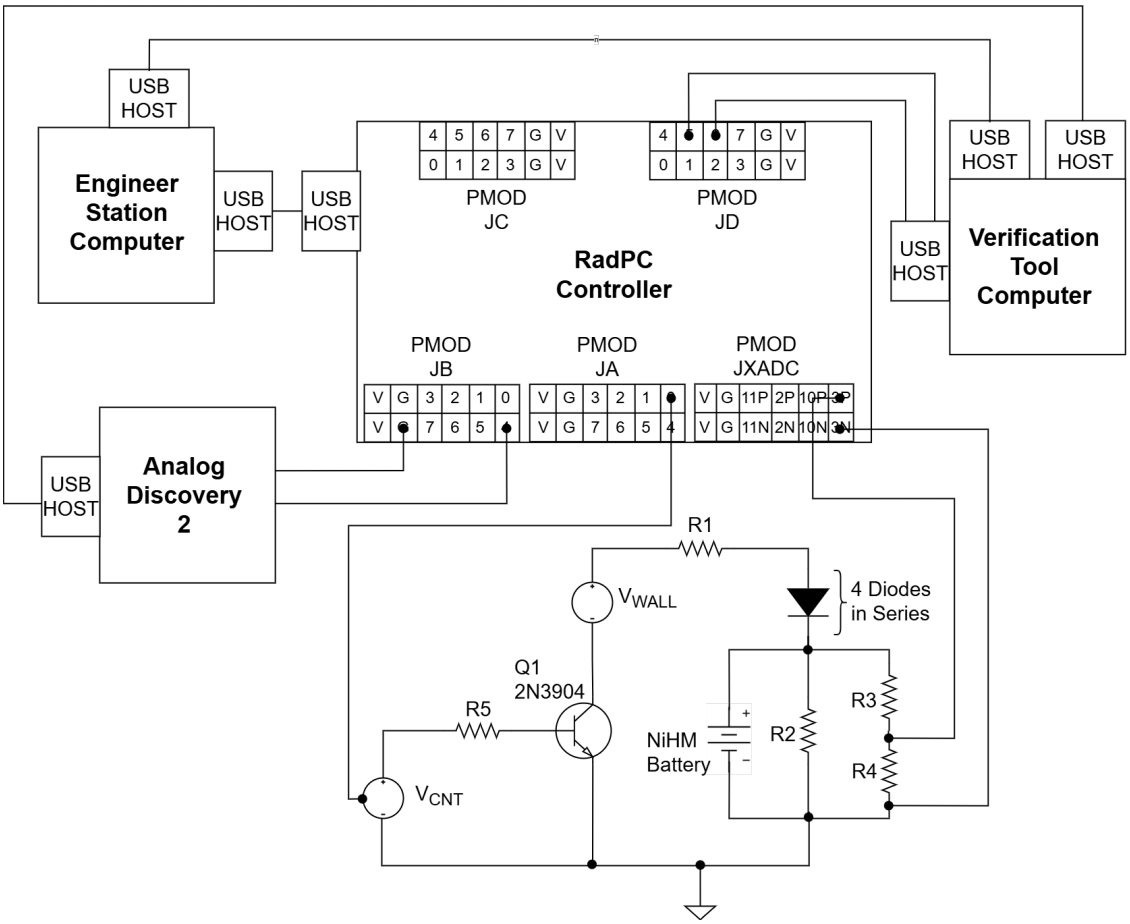

Figure 3: A schematic diagram of the revised process control system (PCS) design. The PCS was redesigned to contain one FPGA with a RadPC core that both controlled the system and monitored voltage. The GPIO of the board consists of the labeled PMOD rectangles, with the individual pins labeled in the squares. Dots on the diagram represent where multiple components are joined. USB connections for communication are designated with a single line. Analog Discovery 2 provides voltage monitoring to confirm the proper performance of the system in tests.

battery state of the charge sensor. It can send and receive data from the engineering station. The State of Charge Sensor block measures the voltage across the battery to provide the current SoC for the battery. The Battery Charger block serves as an actuator, charging or discharging the Battery block based on the feedback logic from the SoC Sensor block.

The external components are an Attacker block, which has access to send and read data from the channel between the engineering station and the FPGA, and a Verification

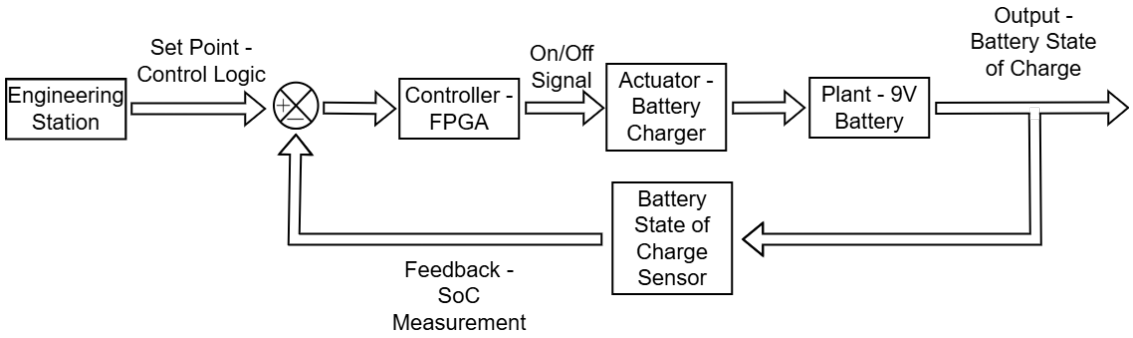

Figure 4: A block diagram of a simple closed-control loop that uses feedback to maintain a setpoint. The loop's set point is control logic that would produce the desired behavior from the system based on feedback measurements. The loop's feedback is the state of charge for the 9 V battery provided by the sensor. The summing point into the controller takes in the set point and feedback.

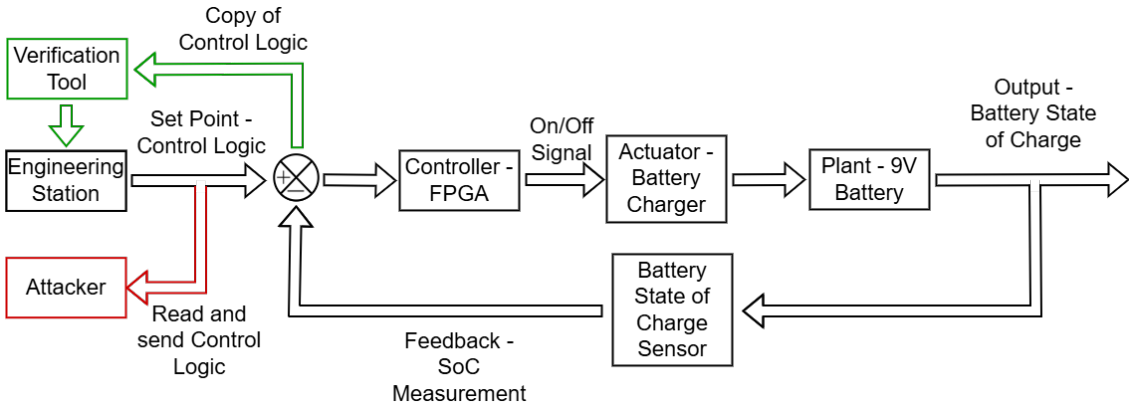

Figure 5: A block diagram of a closed-control loop that uses feedback to maintain a setpoint and includes exterior components that communicate with the control loop. The traditional control block components are lined in black. The exterior components are lined with green if they're trusted and red if they are malicious.

Tool block, which has access to read from the aforementioned channel and send data to the FPGA and engineering station. In this system, the attacker will attempt to send in malicious commands and measurements to force the system to respond in a way that does not align with its current state. For example, higher voltage measurement reports could be sent in when the battery is actually low, resulting in the battery not receiving charge when necessary. Furthermore, attackers could send new logic code into the FPGA, for example, code to always send the off signal to the charger, regardless of the battery's reported SoC.

Recall that Figure 3 shows the connections where the PCS components are connected with wire, indicated with dotted ends, as well as the cable connections between the USB ports of the PCS components.

Circuit Design

To take a closer look at the physical design of the charging circuit, a diagram of the circuit has been included in Figure 6. There are three voltage sources within the circuit: a 12 V nominal voltage wall source, a 9 V battery, and a controlling voltage source for the bipolar junction transistor (BJT), which puts out either 0 or 3.3 V provided by the RadPC FPGA. The 12 V wall source provides 12.3 V in real-world operation. The first portion of the circuit, after the 12 V source, contains four 1N4001 diodes in series and resistor R1. The four diodes drop around 2.8 V altogether and act as a physical fail-safe to ensure a 9 V battery cannot be charged beyond 9.5 V. The resistor R1 is used to control the amount of current supplied to the parallel battery portion of the circuit.

The parallel battery portion of the circuit has three branches. The first branch contains the 9 V battery. The second branch includes a load provided by the R2 resistor. The current for this load can either be supplied by the battery when the circuit is in the discharging state or provided by a 12 V wall source when the circuit is in the charging state. As such, the values of R1 and R2 affected the allowable range of charge voltage for the battery due to the net of their charging and discharging current. The final branch in the parallel portion contains a voltage divider and voltmeter. To make the current through this branch negligible, the resistors are set to exceptionally large values, with R3 set to 750 kΩ and R4 set to 75 kΩ. This divider allows a voltmeter across R3 to read in a value of 1/10 of the 9 V battery's voltage. The voltmeter is provided by an analog-to-digital converter on a Nexys A7 FPGA board, which can read in values of 0 V to 1 V.

To allow the circuit to turn battery charging on and off based on a digital signal from

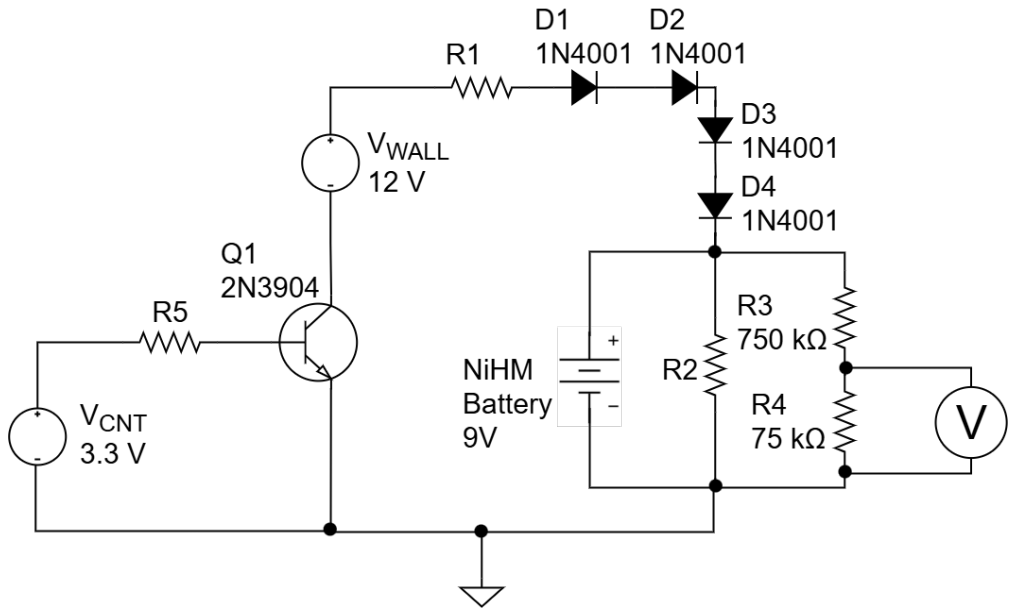

Figure 6: A schematic diagram of the process control system's battery charging circuit. The circuit is designed such that using a control signal the battery can be set to either a charging or discharging state. Additionally, safeguards are present to prevent overcharging of the battery.

the FPGA, a 2N3904 Negative-Positive-Negative (NPN) type BJT transistor is placed before the 12 V wall source and used as a switch. The RadPC FPGA board provides a voltage signal to the base of the BJT, of either 0 V or 3.3 V, and resistor R5 controls the current into its base. The value of R5 is chosen so that current into the base is sufficient for the BJT to enter the saturation region, considering the chart shown in Figure 7 [1]. The RadPC board provides a common ground for the circuit as well, to address issues with floating ground. When a 3.3 V is provided by the board to the BJT's base, it enters the saturation region and acts like a closed switch, allowing the 12 V source to charge the battery. When the board supplies 0 volts, then the BJT is in the cutoff region and acts like an open switch, separating the parallel portion of the circuit from the 12 V source and causing the battery to use up its charge to power the load instead of charging.

Within this base circuit, the resistors R1 and R2 can be swapped to produce different

[1] http://www.secosgmbh.com/datasheet/products/SSMPTransistor/TO-92/2N3904.pdf

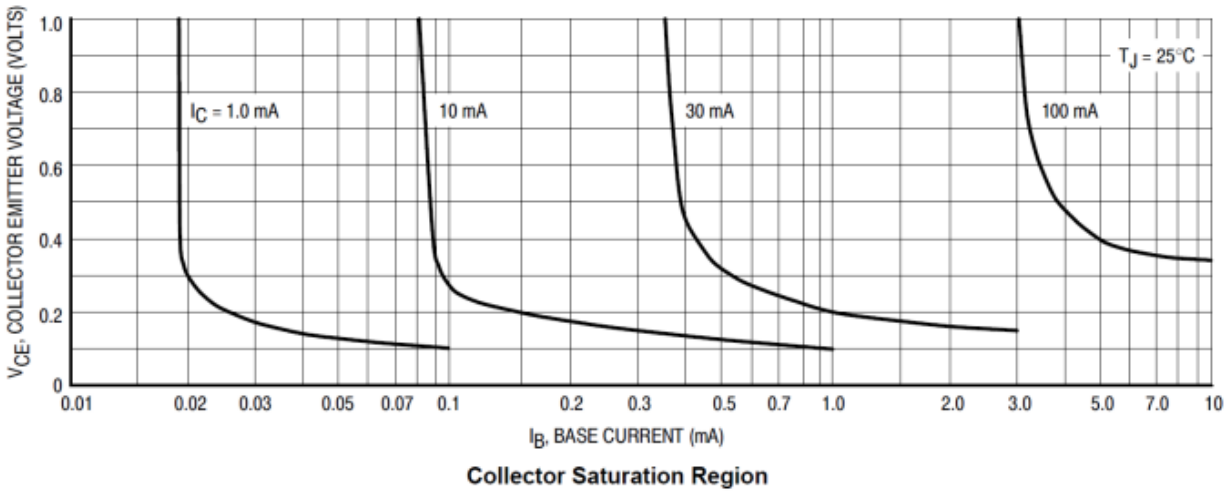

Figure 7: A clipping of the collector saturation region chart taken from the 2N3904 NPN BJT datasheet. This chart was used to choose an R5 value in the circuit such that the base current was sufficient for the BJT to be in the saturation region.

ranges of charge for the battery. Two sets of values were designed. The first set R1 to 68 Ω and set R2 to 500 Ω, which physically allowed for a lower voltage operation range of 5.2 V to 6.9 V. The second set R1 to 22 Ω and set R2 to 750 Ω, which physically allowed for a higher voltage operation range of 6.5 V to 8.0 V. The charge rate of the battery provided by these values fluctuates based on the current voltage of the battery. Graphs have been included in Figures 8, 9, 10, and 11, mapping the full discharge and charge of the battery using the two resistor combos.

Engineering Station Design

The Engineering Station was designed as a Python file in Visual Studio Code and executed on a Ubuntu Linux operating system. The engineer can use the program to either (1) send in a value that will change the desired range of operation for the charging system or (2) send in a predetermined bitstream to bootload the logic code on the RadPC controller. The program allows the engineer to choose to set either the maximum voltage or the minimum

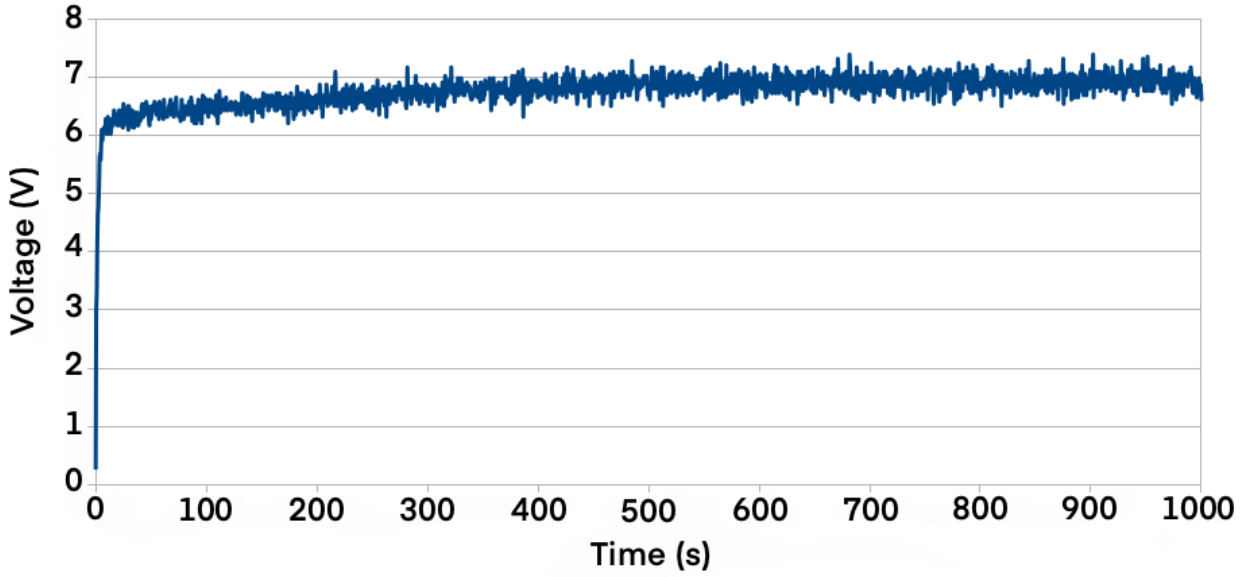

Figure 8: A graph of the 9 V battery's voltage as the process control system takes it from depleted to charged. In this case, the chosen values of R1 and R2 resulted in a lower maximum voltage than the resistor values used in Figure 9.

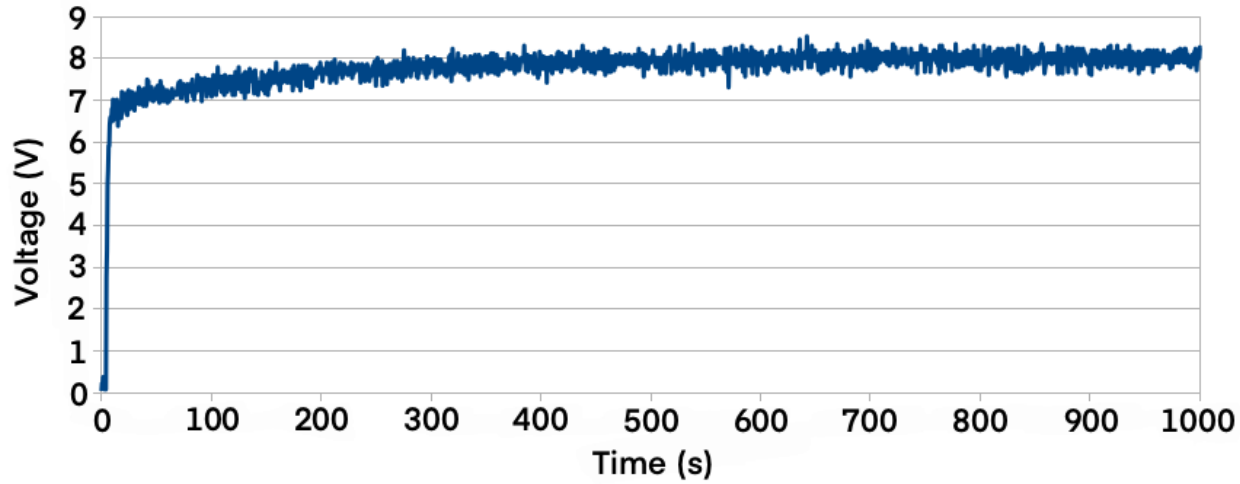

Figure 9: A graph of the 9 V battery's voltage as the process control system takes it from depleted to charged. In this case, the chosen values of R1 and R2 resulted in a higher maximum voltage than the resistor values used in Figure 8.

voltage, allowing for precision up to the first decimal point. The maximum voltage is only allowed to be set up to 9.5 V and the minimum voltage must be greater than 1.0 V. Additionally, the maximum voltage must always be set to greater than the minimum voltage

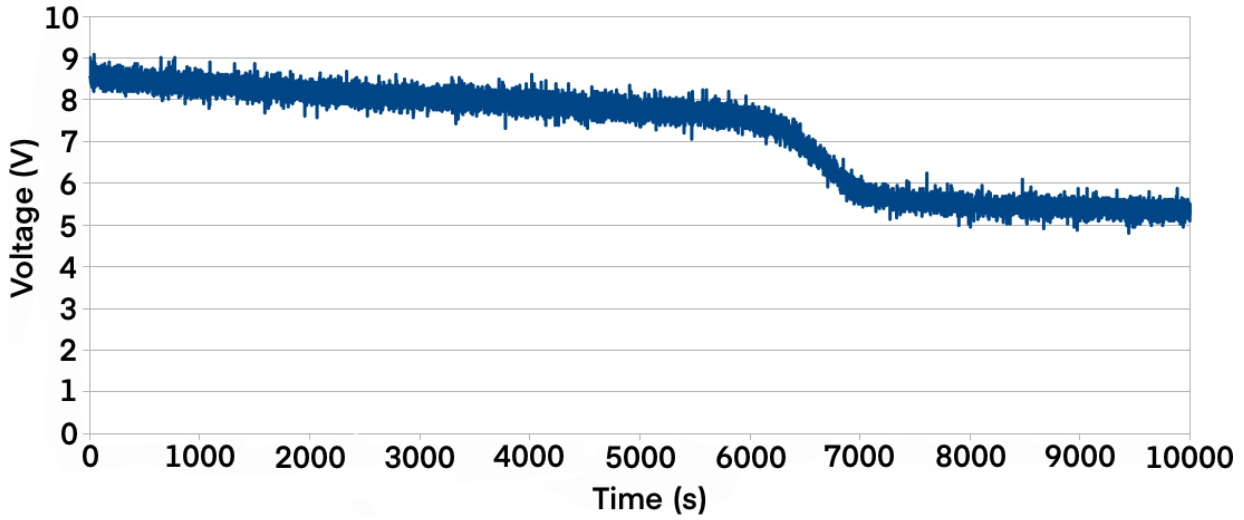

Figure 10: A graph of the 9 V battery's voltage as the process control system takes it from charged to depleted. In this case, the chosen values of R1 and R2 resulted in a lower minimum voltage than the resistor values used in Figure 11.

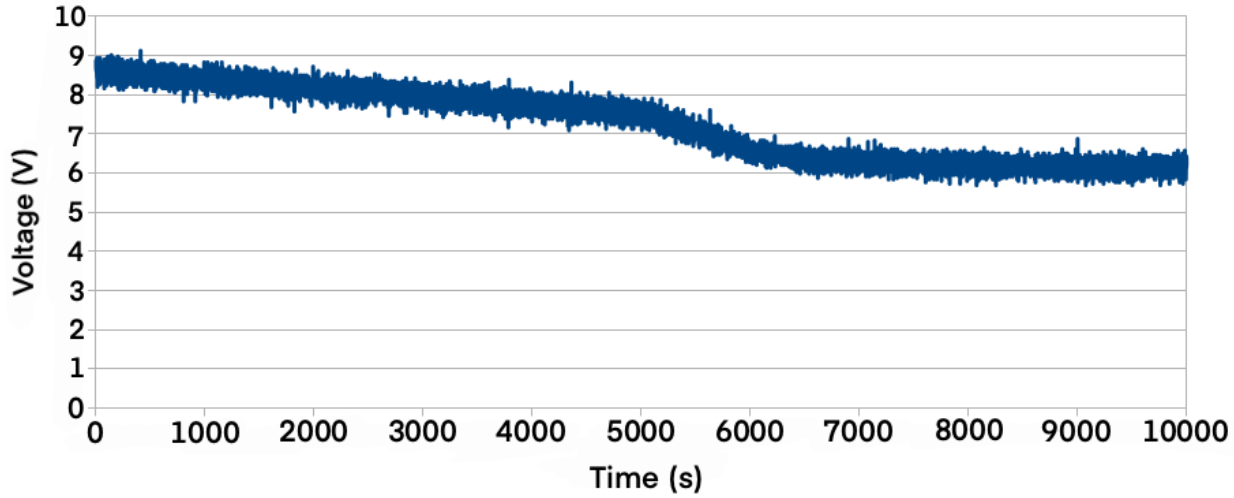

Figure 11: A graph of the 9 V battery's voltage as the process control system takes it from charged to depleted. In this case, the chosen values of R1 and R2 resulted in a higher minimum voltage than the resistor values used in Figure 10

and the minimum to less than the maximum. For any commands or firmware files being sent, the engineering station program tacks on the uncommon byte *'\x7f'*, a hex representation of Delete, to the data just before the terminating *'\n'*. This allows the verification tool to identify the end of individual transmissions to RadPC. In cases where the program is not

being actively used by an engineer, a heartbeat command is sent out to keep RadPC's logic code from stalling, as RadPC does not currently support timeout logic for receiving data.

Additionally, the engineering station is always listening for possible attack reporting from the verification tool. Multithreading was implemented to allow the station to receive and process this data concurrently with sending data. Multithreading describes the ability of a processor to run multiple threads, each performing different functions separately but in quickly interleaving turns, such that they appear to run simultaneously. With the use of threading, if one thread is waiting to send or receive, the other thread can send or receive while the first thread is inactive.

Field-Programmable Gate Array Logic

A base level of logic code for the board was designed as bitstreams for RadPC, synthesized from hardware description language (HDL) files in Vivado. Vivado was also used to upload the resulting bitstreams to both boards. The logic code was developed from the base HDL files provided for the boards by the Resilient Computing lab for RadPC. Alterations include altered logic to control the flow of UART traffic that is received by RadPC's two UART ports, logic to output an on or off signal for the charger on a GPIO pin and led, and logic to output an on or off signal for an attack indicator on another GPIO pin and led.

Alterations of UART traffic flow to RadPC varied for the different versions of the verification tool. For the detection version of the tool, all signals sent to the FPGA from the engineering station are received by the RX line of the primary UART and saved to its RX register as normal. The signals are additionally saved to the secondary UART's TX register to be forwarded to the verification tool through its TX line. For the mitigation version of the tool, all signals being sent to the FPGA from the engineering station are received by the RX line of the primary UART, but not saved to its RX register. The signals are instead

saved to the secondary UART's TX register to be forwarded to the verification tool through its TX line. Lastly, the RX register of the primary UART saves data received by the RX line of the secondary UART from the verification tool. This flow of UART traffic is necessary, as it simulates the passive serial tap used by the verification tool. It also allows the secondary UART to send control logic to be bootloaded onto the board, which only allows control logic saved to the primary UART's RX register to be bootloaded.

A secondary, higher level of logic code was designed as a C file in Visual Studio Code and executed on the RadPC board. The code was compiled using RadPC's software workflow environment and sent over a UART cable to be bootloaded on the RadPC board. The logic code provides the operating control logic for the PCS based on feedback measurements. Every time heartbeat commands are received from the engineering station or verification tool, it samples a 12-bit resolution binary value read in from RadPC's Analog-to-Digital-Converter, which reads across the battery's voltage, and the logic converts it to a decimal representation that includes a tens place digit, a ones place digit, and a tenths place digit. Using this decimal representation, the current decimal voltage value is checked against the desired maximum and minimum voltage value limits. It turns the BJT base control signal off or on accordingly by altering the GPIO value provided to the I/O pin.

If the charger signal is off and the current voltage is below the minimum voltage, the signal will turn on. If the charger signal is on and the current voltage is any value below the maximum voltage, the charger will remain on. In contrast, if the charger signal is off and the current voltage is anywhere above the minimum value, the charger will stay off. Lastly, if the charger signal is off and the current voltage is any value below the minimum voltage, the charger will be turned on. This results in a system that can bounce back and forth between the minimum and maximum charges to maintain a desired range of operation, usually outside the battery's range of depletion, to improve performance.

Attacker Design

Attackers in ICS commonly seek to achieve the Impair Process Control tactic, MITRE ID TA0106, with the end goal of affecting the availability of a system. If the battery charger PCS was part of a larger ICS and the role of the battery was to provide power to an ICS component, attackers could prevent the expected performance of the ICS by attacking the battery PCS. For example, an attacker may provide unauthorized command messages to the FPGA, such as improper maximum or minimum voltage limits or their harmful bitstream, which could instruct the charger signal to be off when it should be on or vice versa. Thus, these attacks could result in a dead battery or an overcharged battery, achieving the Impair Process Control tactic.

For ease of use in this system, the attacker is implemented as a separate functionality within the Python engineering station program to allow access to the communication channel between the station and the RadPC controller. This allows the attacker to send in both malicious commands and malicious logic code. Some malicious commands include providing illogical voltage limits, which are too high, too low, or contrary to each other. The malicious logic code is sent over as a compiled firmware file. For these attacks, it is assumed that the attacker may be aware of the proper forms for commands, including the *'\x7f'* terminating byte.

Verification Tool Design

The verification tool for the system was designed as a Python file in Visual Studio Code and executed on an Ubuntu Linux operating system. Ideally, the verification tool is run on a separate computer, so it is separate from the engineering station network, but in practice, it may be run on the same computer. The verification tool is also connected to the engineering station so that it can send in reports on detected attacks. The verification tool forwards

data received by RadPC's primary UART port through hardware functionality. This is done by forwarding the data to RadPC's secondary UART port, which is connected to the verification tool host using an FTDI Chip C232HD-DDHSP-0 USB to UART cable [11]. The connections of this UART cable are shown in Figure 12. This setup was implemented to behave similarly to a passive serial communication tap between the engineering station and the RadPC FPGA.

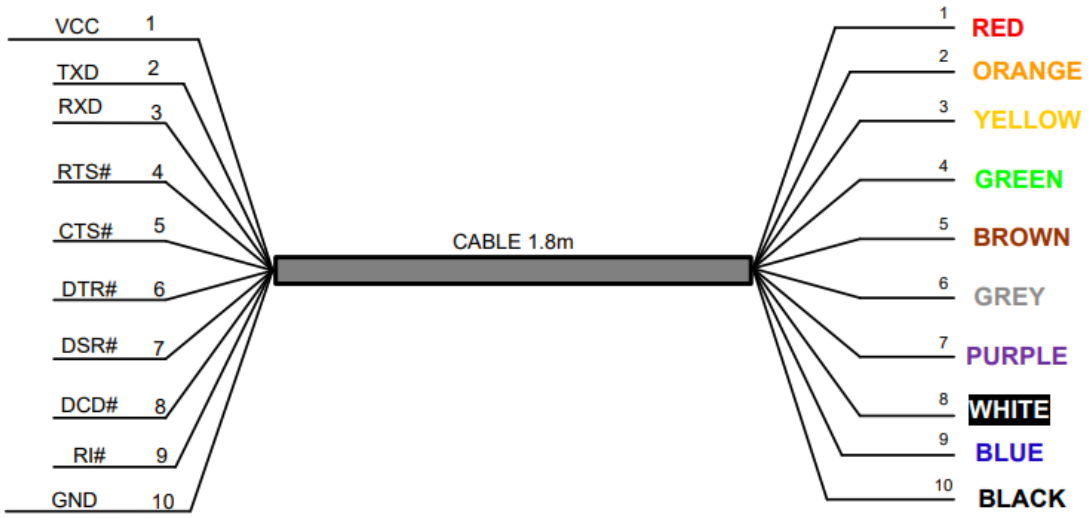

Figure 12: A clipping of the cable connection chart from the FTDI Chip C232HD USB to UART Cable Datasheet. This cable was used to physically wire the passive serial tap between the process control system controller and the verification tool computer.

The verification computer is then able to receive the incoming serial transmission and transmit it to the verification tool. The serial port for communication was configured to the serial data requirements of the RadPC, specifying a baud rate of 115200, 8-bit data, no parity, and no flow control. Prior to being connected to the computer, an Analog Discovery was used to confirm the UART signal being forwarded. When receiving data, the verification tool looks for the terminating bytes *'\x7f'* to separate individual transmissions.

There was base functionality for the verification tool that applies to all versions. All versions of the verification tool look for expected transmissions to be sent by the engineering

station and validate any incoming transmissions against the expected transmission. If any anomalous transmissions are detected, they are logged. It focuses on verifying commands and logic code rather than firmware. It verifies that any engineer commands being sent to the board match up to the pre-approved good commands. Additionally, for any commands sent in that follow the correct format to change the voltage limits, it verifies the new voltage limit being set. To do this, the verification tool keeps track of the current set maximum and minimum voltage limits and checks that the voltage value fits the same rules required by the engineering station input program. If any commands or voltage values don't agree with expected good values, they are saved to a log file *invalid_ command_ log.text*. This file provides part of the tool's detection functionality.

Mitigation for DoS attacks was also focused on, as it posed a potential problem with serial communication used by the verification tool. To address this, the bounds of the information being sent were considered, and it was identified that the longest a potential set of data would be was the size of the trusted firmware file in bytes, plus one byte for the terminating byte. Thus, the trusted firmware size was chosen as a limit for the serial reading function, only allowing it to read up to that number of characters before processing data.

Additionally, the verification tool also verifies incoming logic code. It does so by recording incoming bitstreams and performing checksums on them. The captured bitstream is checked against a predetermined good bitstream compiled from a pre-approved logic code. Specifically, the two files are verified using a process with the SHA-256 algorithm. The two files are both hashed, and checking is done to see if the two hashes are equal. If the incoming hash is not equal to the good hash, it is permanently saved to a log file *invalid_ bin_ log.text*. This file provides another part of the tool's detection functionality. Multi-threading was implemented to allow the tool to receive and process data while sending back information at the same time. The engineering station, logic code, and firmware perform precisely as described in their component sections.

Two versions of the verification tool were implemented on the base design described above. The first version provided only passive detection, monitoring traffic being sent to execute in the FPGA and reporting if an attack was detected on its computer. The final version provides both reactive detection and prevention, requiring verification of traffic sent to the FPGA before it is sent back in for execution and reporting attacks back to the engineering station. To support these different versions, the code for the engineering station and the logic code for the FPGA have slight variations.

Version 1: Passive Detection

For the passive detection version, if transmissions outside the expected are being received, the verification tool logs the offending transmissions and alerts to a possible attack on its computer. Additionally, the tool reports information on the attack back to the engineering station. This provides passive detection that includes reporting for potential attacks but does not provide any prevention against detected attacks besides the base protection against DoS attacks. No functionality of note has been added to this code beyond that of the base functionality. Additionally, the FPGA turns on an LED to indicate that an attack occurred. However, similarly to the first version, the tool does not provide any prevention against detected attacks besides DoS.

Version 2: Detection with Mitigation

For the final version of the verification tool, prevention functionality was added to the detection functionality of the first version. In practice, prevention is typically a combination of the detection of attacks and a response to prevent the effects of attacks. Upon researching the topic, it seems PCS verification tools are focused primarily on detection rather than prevention, so this was included to further expand upon the subject. There are slight differences in the communication processing for components in this version of the verification tool.

In this version, the FPGA cannot receive commands directly from the engineering station and instead receives them from the verification tool. The verification tool forwarded commands from the engineering station instead, which it verifies before allowing those commands to be forwarded and executed on the FPGA. To the eyes of the engineering station, however, commands still appear to be directly sent from itself to the FPGA's primary UART without any interference. As the FPGA no longer looks to the engineering station for its commands, the engineering station program no longer sends a heartbeat signal. Instead, the verification tool provides a heartbeat signal in the absence of other commands to send to the FPGA.

RESULTS

Experimental data yielded promising results, with the verification tool successfully providing prevention and detection for the FPGA controller against unauthorized command message attacks on the serial communication channel between the engineering station and FPGA. We will explore the results of this data in the following sections.

Normal Operation

A baseline of normal operation of the PCS was established by using the probes of an Analog Discovery 2, which reported voltage values back to a Python program. Successful maintenance of battery voltage by the PCS for both resistor combos is displayed in voltage charts, which display the battery voltage in blue and the control signal of the charger in orange, with high meaning the charger is on and low meaning it is off. For the lower voltage range shown in Figure 13, the maximum and minimum voltage limits were changed every 700 seconds, starting at voltage limits of 5.2 V to 6.9 V, then 5.4 V to 6.7 V, and finally ending at 5.6 V to 6.5 V. For the higher voltage range shown in Figure 14, the maximum and minimum voltage limits were changed every 2000 seconds, starting at voltage limits of 6.5 V to 8.0 V, then 6.8 V to 7.7 V, and finally ending at 7.0 V to 7.5 V.

It was also confirmed that the FPGA could be successfully bootloaded with the trusted firmware file when the engineering station requested. Figure 15 shows the normal operation of the engineering station, the verification tool, and the FPGA. It shows the engineer sending in a low voltage limit of 1.3 V and a high voltage limit of 9.3 V, as well as the detection of valid commands by the verification tool. Lastly, it shows successful updating of the voltage limits and the charger control signal on the FPGA, which displays the current voltage in volts, the maximum voltage limit in volts, the minimum voltage limit in volts, and the charger control signal value on its seven-segment display.

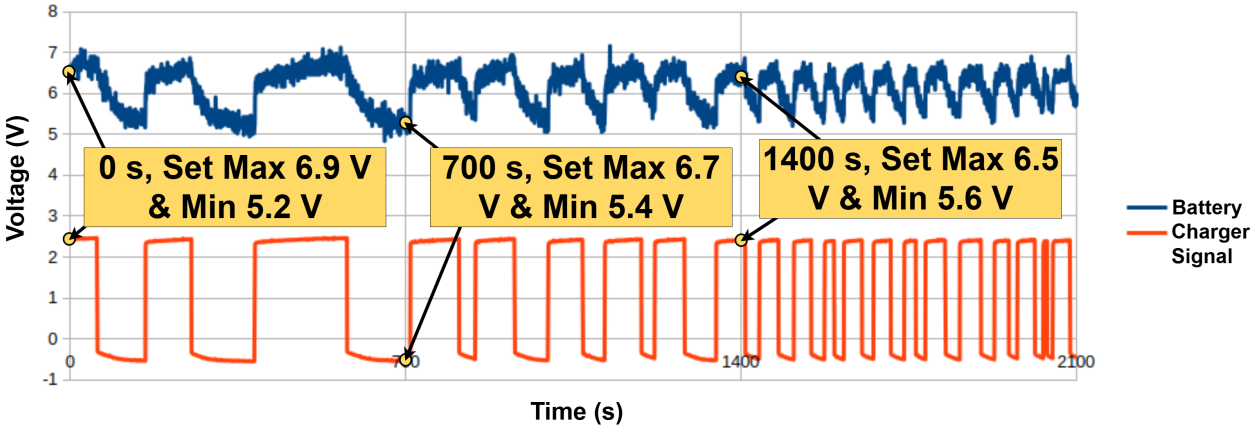

Figure 13: A graph that shows maintenance of the battery's voltage limits for three different ranges by the process control system (PCS) with resistor values chosen for R1 and R2 that result in a lower permittable voltage range than in Figure 14. In this graph, we see that the battery voltage begins to charge or discharge in correspondence with the PCS's charger control signal, confirming the expected performance of the system.

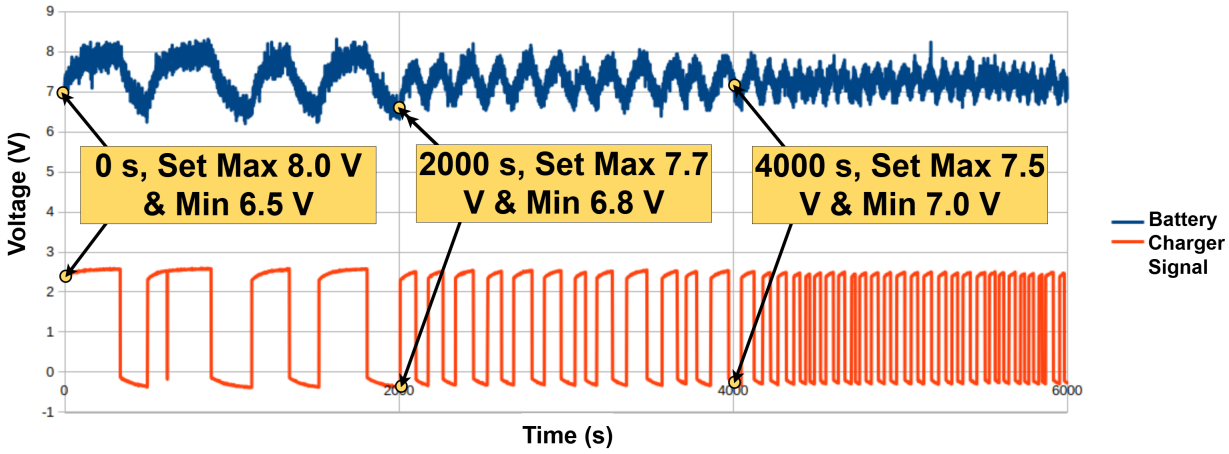

Figure 14: A graph that shows maintenance of the battery's voltage limits for three different ranges by the process control system (PCS) with resistor values chosen for R1 and R2 that result in a higher permittable voltage range than in Figure 13. In this graph we see that the battery voltage begins to charge or discharge in correspondence with the PCS's charger control signal, confirming expected performance of the system.

Detection

Three attack cases were sent into the FPGA by the attacker: two voltage setting commands that went out of bounds, one for high and one for low, and a harmful firmware

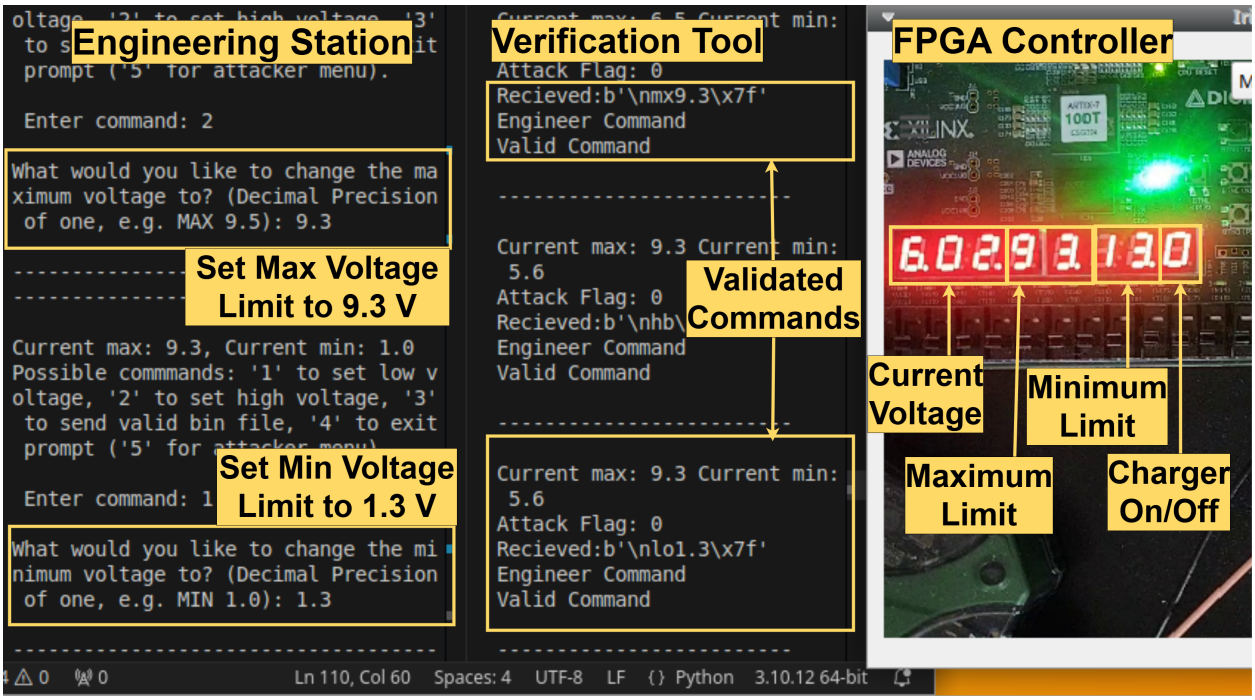

Figure 15: The process control system testbed under normal operation. User interfaces are shown for the engineering station and the verification tool. The physical controller is also shown, and its seven-segment display shows relevant information for the system. These displayed interfaces show the engineer sent in new maximum and minimum limits for voltage maintenance and that the values across all three displays were correctly updated.

file. In all these cases, the verification tool was able to passively detect and log the possible attacks. For the tool, successful detection was denoted with a message on its commands window, recording to its log file and reporting to the engineer station. Both versions were able to successfully inform the FPGA and the engineering station of the attack. This reporting was given as a message being sent back to the engineering station and an indicator LED being lit on the FPGA.

Detection was also achieved in all three attack cases. An example was recorded to show the system's response to out-of-bounds limit attacks. For this case, the maximum limit was only allowed to be set up to 6.5 V, and the minimum voltage was only allowed to be set as low as 5.6 V, and the lower range resistor combo was used. The attacker sent

an unauthorized command message that set the max voltage limit to 9.9 V, well above the allowable limit. The resulting graph of battery voltage and the charger control signal is shown in Figure 16. The system ran normally for the first 60 seconds when the attacker sent in the harmful voltage limit. Because our system has physical limiters for the maximum voltage of the battery in the form of dropped voltage by diodes and will only charge as long as the current into the battery is greater than the current provided to the load, this resulted in the battery never being able to hit that maximum limit and discharge again. In a system without a physical limiter on the battery's charge, this may have resulted in an overcharged battery. Figure 17 shows the response of the physical PCS and its user interfaces to the attack. A detection report is visible on the user interfaces of the engineering station and the verification tool. On the FPGA display, we can see that the maximum limit has been updated to 9.9 V, which is above the maximum limit normally allowed by the system.

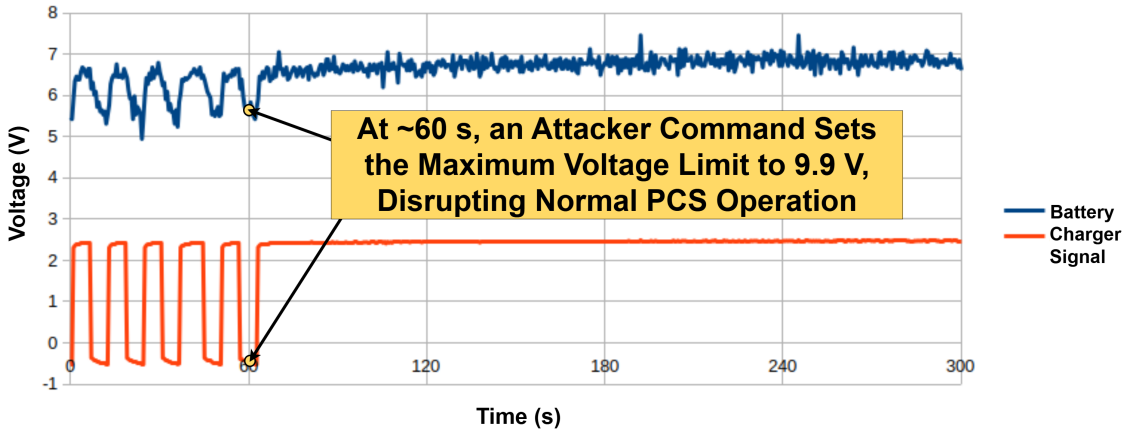

Figure 16: The graphed response of the process control system to an unauthorized command attack that sets its maximum voltage limit to 9.9 V. In this case, the verification tool which provides only detection was used, so the attack was not prevented. In its response, we see that the normal behavior of the system is disrupted. It no longer maintains the battery's voltage between the two expected limits. Instead, the charge signal remains on indefinitely, and the battery voltage stays in a charging state. In the designed process control system, the battery will never reach 9.9 V, so the battery will remain physically unharmed. Still, in the system without a physical limiter, this may result in overcharging of the battery.

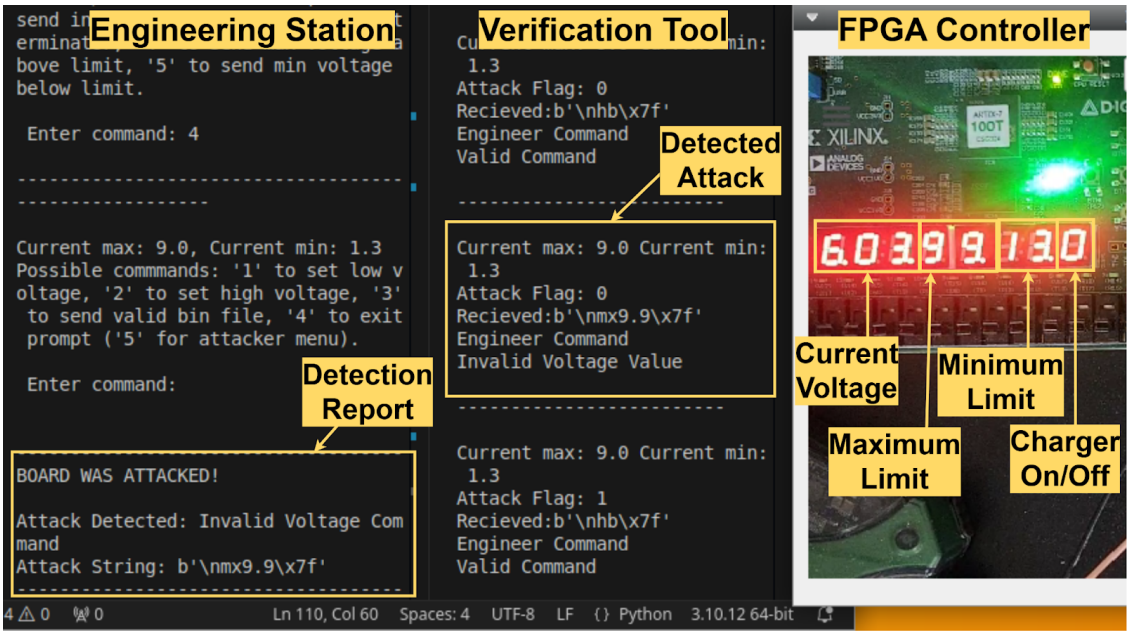

Figure 17: The response of the physical process control system and its user interfaces to an unauthorized command attack that sets its maximum voltage limit to 9.9 V. In this case, the verification tool which provides only detection was used, so the attack was not prevented. A detection report is visible on the user interfaces of the engineering station and the verification tool. On the FPGA display, we can see that the maximum limit has been updated to 9.9 V, which is above the maximum limit normally allowed by the system.

Prevention

Beyond detection, successful results were also achieved for prevention, for which the attack needed to be addressed before reaching the FPGA. The second version of the verification tool was able to prevent all three of these attacks from reaching the FPGA controller while allowing the system to perform as if the verification tool was not there. In all cases, it was able to identify that the data being sent in belonged to a given attack and prevented it from being forwarded. The maintenance of SoC for the battery was once again monitored, and it behaved precisely according to the expected maintenance performance. Figure 18 shows the experimental setup after malicious firmware has been sent by the attacker. It shows the initial detection by the verification tool, the notification of the attack

report by the engineering tool, and the unaffected firmware performing on the FPGA. Figure 19 shows the experimental setup after malicious voltage limits have been sent by the attacker, with values of 0.5 V for the minimum limit and 9.9 V for the maximum limit. It shows the initial detections by the verification tool, the receiving notifications of the attacks report by the engineering tool, and the unaffected voltage limits on the FPGA.

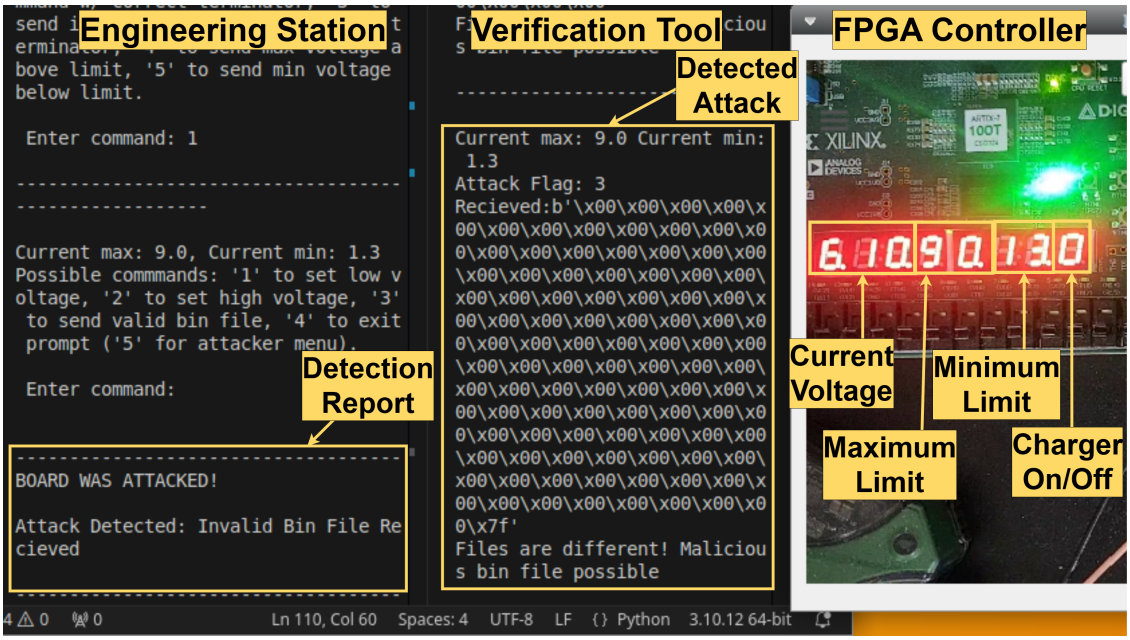

Figure 18: This figure confirms the successful detection and mitigation of an attempted malicious logic code upload on the process control system by the detection with prevention version of the verification tool. Detection of the attack is reported on the user interface of both the verification tool and the engineering station. We also see that the normal program remains running on the FPGA controller, so the malicious logic code was prevented from bootloading on the controller.

Evaluation of Performance

This provides details of the evaluations of the battery charging PCS's performance, using CSV file records and graphs of the system's battery voltage and charger control signal voltage. Multiple instances of voltage overshoot from the battery were observed, where the voltage went above the upper limit and below the lower limits, as shown in the charts of

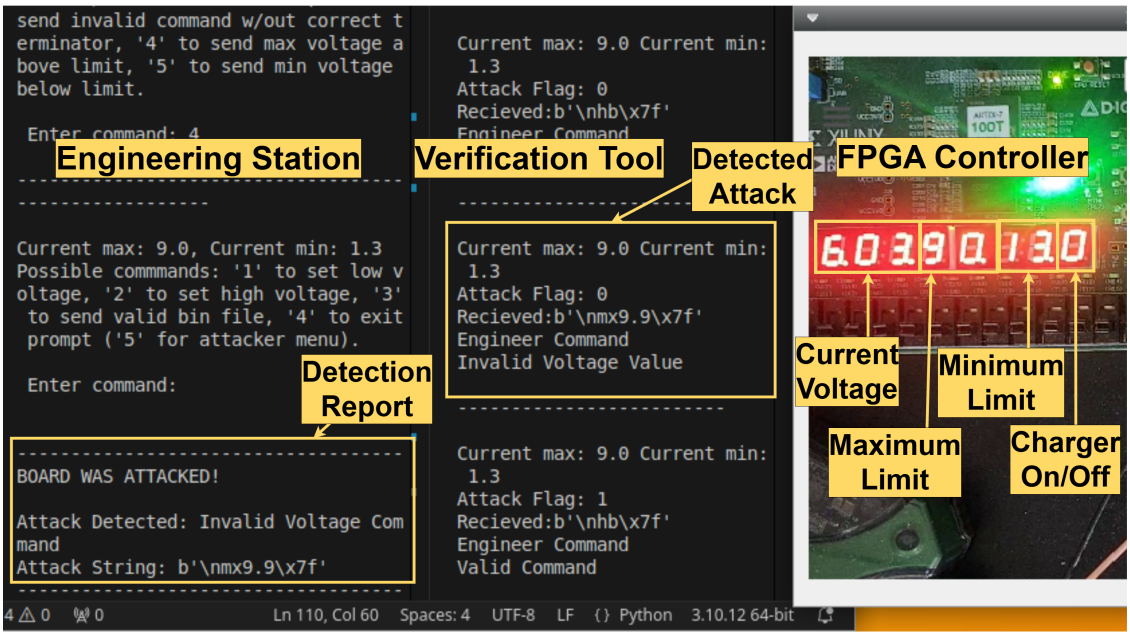

Figure 19: This figure confirms the successful detection and mitigation of an attack which sends in a malicious maximum voltage limit. Detection of the attack is reported on the user interface of both the verification tool and the engineering station. We also see that the malicious maximum limit is not set on the FPGA controller, because it was prevented from being set in the controller.

Figures 13 and 14, were observed. In control systems, overshoot is the behavior where the actual output of the system exceeds the target value. In part due to this overshooting, the elapsed time of charge-discharge cycles was varied. The performance of the PCS was recorded using voltage charts. Figure 21 shows the recording of the performance of the system without the verification tool, and Figure 20 shows the recording of the performance with the verification tool. For the first voltage range, each system experienced about four charge-discharge cycles within the first 200 seconds. After 200 seconds, when the voltage range was updated, the system without the tool experienced 18 charge-discharge cycles, and the system with the tool experienced 15 cycles. Considering the present variation of the charge-discharge cycle, the difference in performance time for the tool appears negligible. This is confirmed in the next section using statistical analysis.

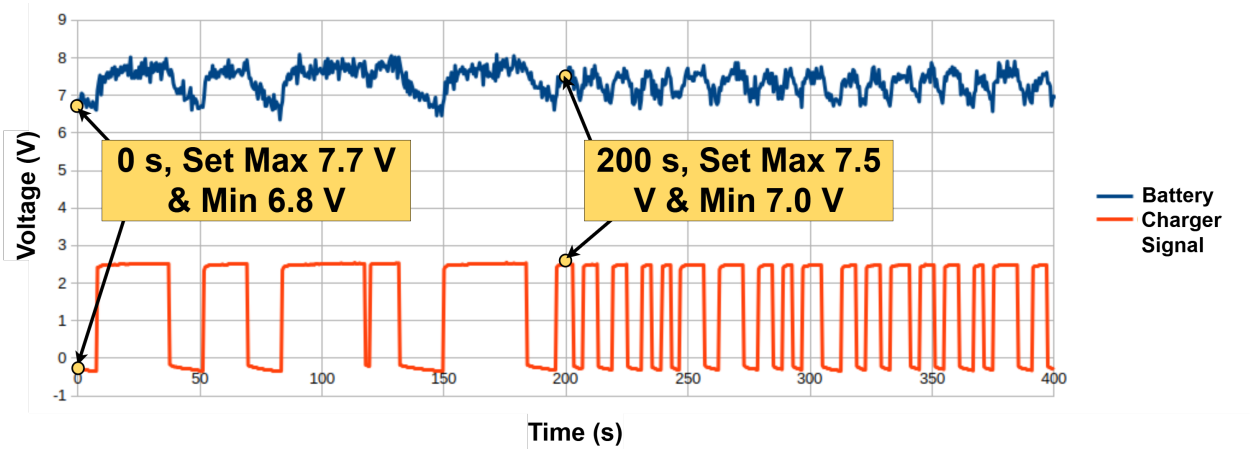

Figure 20: This figure graphs the battery voltage and charger signal voltage for the battery charging PCS without the verification tool. At the start, the maximum voltage limit is 7.7 V and the minimum limit is 6.8 V. After 200 seconds, the limits are updated to be 7.5 V and 7.0 V. The system response and performance is visibly similar to that of the chart in Figure 20.

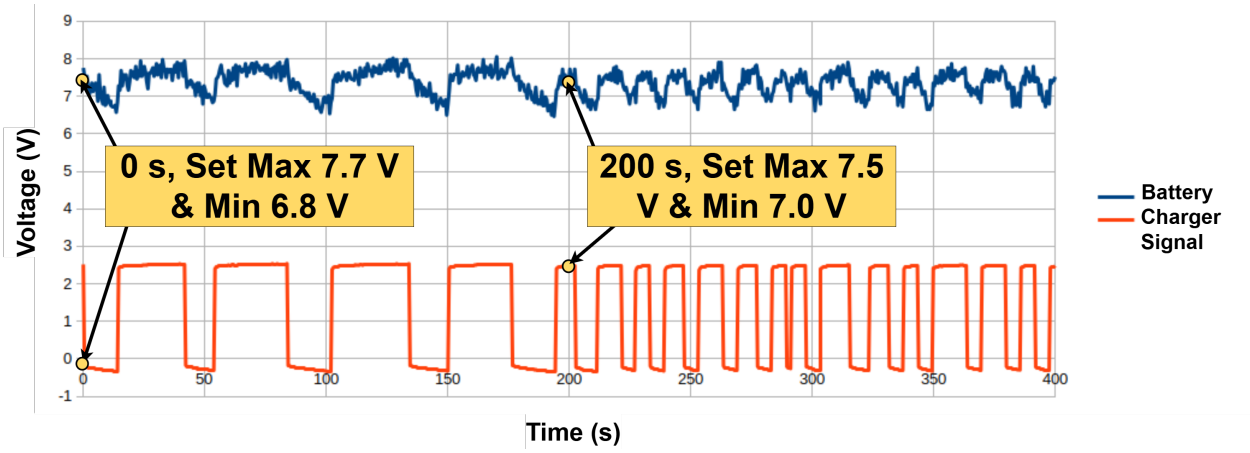

Figure 21: This figure graphs the battery voltage and charger signal voltage for the battery charging PCS with the verification tool. At the start, the maximum voltage limit is 7.7 V and the minimum limit is 6.8 V. After 200 seconds, the limits are updated to be 7.5 V and 7.0 V. The system response and performance is visibly similar to that of the chart in Figure 21.

Statistical Analysis of Performance

The tables in Figure 22 on this page show the results obtained from running statistical analysis on PCS performance data using the R programming language. For the specific code

used, please refer to the appendix. Statistical tests were performed on data showing the performance for the PCS without a verification tool and the PCS with the verification tool, and with both the systems maintaining the battery voltage at 6.8 V to 7.7 V for the first 200 seconds and then 7.0 V to 7.5 V for the last 200 seconds. There were four columns in the combined data, indicating: inclusion or exclusion of the tool, the runtime of the system in seconds, the voltage of the BJT control signal in volts, and the voltage of the battery in volts. I performed two tailed multiple linear regression analysis on the PCS performance data. The null hypothesis of the test is that the performance of the PCS which uses the verification tool is equivalent to the performance of the PCS not using the verification tool.

I chose this test because linear regression is optimal for continuous independent variables and I had multiple independent variables. Two-tailed analysis provides a measure of how much evidence there is to reject the null hypothesis. The test has three independent variables which affect the value of the battery's voltage: the time in seconds is a continuous variable, the inclusion or exclusion of the verification tool is given as a numerical 0 or 1, and the BJT control voltage is a continuous variable. The dependent variable that is affected by the independent variables, battery voltage, is continuous as well. Multiple Repeated Anova testing was also considered for this analysis, but wasn't used because the independent time variable is continuous, and anova works best with class based independent variables.

When analyzing the dataset, the p-value for the "Has Tool" variable was much greater than 0.05, and thus did not show statistical significance to the dependent value of the battery voltage. This is in agreement with the null hypothesis and thus suggests that the performance of the system with the verification tool is similar to the performance of the system without the verification tool, so the tool can be used in the system without affecting performance. On the other hand, the p-values for "Time" and "Control Voltage" were much smaller than 0.05, which suggests that they are statistically significant to the dependent value of the battery value. This is as expected, since the battery voltage is known to change with time

and in accordance to the control signal, which turns the charger on and off. I also performed simple linear regression on each of the independent variables separately, to confirm their likely effects on the voltage without any sway from the other independent variables. The results of the simple linear regressions, based on their p-values, were in agreement with the findings of the multiple linear regression analysis. Overall, these findings suggest that the presence of my verification tool does not have a significant effect on the performance of the process control system, and thus the tool can be included without affecting performance.

Results Discussion

All research goals of the project were met. The verification tool was shown to provide mitigation as well as detection. The PCS featuring the verification tool was able to update its voltage limits and bootload the trusted firmware to the FPGA when requested, just as the original version of the PCS was able to. This suggests availability, when availability is defined for PCSs as *the probability that a system is up and running as expected at some point in time*. When including the verification tool, the execution time of the control system was negligibly impacted, as shown in the Figure 20 and 21 voltage charts. Thus, the tool was added to the PCS without negatively affecting the performance of the PCS, when considering performance as the the time for command execution and the correct execution of commands.

Threats to Validity

Three types of threats to validity were explored in this research, which was grounded on the classification scheme of Campbell & Cook (1979) [14] and Campbell & Stanley (2015) [8]. These threat types include (1) internal threats to validity, which refer to undesired relationships, specifically regarding the extent of confidence that independent variables cause effects on dependent variables; (2) construct threats to validity, which refer to how well a

Multiple Linear Regression

	Coefficients	*Standard Error*	*t Value*	*P-value*
Intercept	7.1517686	0.0138489	516.416	< 2e-16
Has Tool	0.0029636	0.0110002	0.269	0.788
Time (s)	-0.000248	0.0000477	-5.2	2.25E-07
CTL Voltage (V)	0.1869063	0.0040796	45.814	< 2e-16

Has Tool Linear Regression

	Coefficients	*Standard Error*	*t Value*	*P-value*
Intercept	7.36701	0.011928	617.63	<2e-16
Has Tool	-0.007352	0.016884	-0.435	0.663

Time Linear Regression

	Coefficients	*Standard Error*	*t Value*	*P-value*
Intercept	7.44E+00	1.67E-02	444.89	<2e-16
Time (s)	-3.83E-04	7.25E-05	-5.29	1.40E-07

CTL Voltage Linear Regression

	Coefficients	*Standard Error*	*t Value*	*P-value*
Intercept	7.1019	0.007950	893.33	<2e-16
CTL Voltage (V)	0.188196	0.004103	45.87	<2e-16

Figure 22: Tables containing linear regression results. The top table contains results for multiple linear regression performed on the entire process control system performance data and the lower three contain simple linear regression results for its individual independent variables. Overall, the results in the table suggested that the inclusion of verification tool was not statistically significant to the performance of the process control system, while time and BJT control voltage were significant.

study's tests and measurements reflect the real-world concepts the study was designed for, and (3) external threats to validity, which describe the degree to which findings can be generalized.

Internal Validity

In this study, internal validity refers to the certainty that the verification tool was the direct cause of the observed and desired PCS behavior. A recognized threat to internal

validity in this study was in its instrumentation, as the core of the RadPC computer was updated several times throughout its design. This may result in unintended inconsistencies in the resulting system performance.

Construct Validity

In this study, construct validity refers to how well the verification tool worked to detect and prevent attacks without negatively affecting PCS performance. The construct validity of this research was explored by observing the various user interfaces of the PCS and recording the voltage charts of the PCS's battery voltage and charger control signal to ensure proper functionality.

It is recognized that a threat to the internal validity of this project includes the small sample size of attacks that were used on the system. Real-world systems will be subject to a wide range of attacks, and the tool on this study focused only on unauthorized command message and DoS attacks. In future work, testing a wider variety of attacks could result in a more robust tool. Additionally, no system evaluation has been performed that simulates scaling to a larger system, with factors such as heavier communication loads, multiple controllers, or noise in serial communication.

External Validity

In this study, external validity refers to how well the study can be generalized for real-world PCSs. For threats to external validity, it is recognized that the tool was designed for an FPGA-controlled PCS, and its implementation on a generalized PCS using a proprietary PLC instead may prove more difficult. Although the tool could be easily added to a PLC that uses serial communication, such as RS-232, using a physical passive serial tap, it may not be well suited to handle controllers that use different communication methods, such as Ethernet. To mitigate this threat, future development of the tool could be attempted on a proprietary controller or a simulator of one.

CONCLUSION

In conclusion, this paper completed the goal of developing a verification tool that provided detection and prevention against attacks without negatively impacting the performance of its PCS. In particular, this tool focused on detecting and preventing attacks using unauthorized command message techniques on its RISC-V FPGA-controlled PCS. The most noteworthy contribution is the development of this mitigation functionality for the verification tool, as this functionality may be applied to verification tools for other standard PCS controllers that had focused solely on detection previously. Because the tool only requires the hardware inclusion of a passive serial tap to a system to monitor its traffic, it would be physically easy to incorporate this system into a PCS. However, rerouting communication to pass through the verification tool before it is forwarded to the controller may prove more difficult. Further development of this tool can be explored by expanding its functionality for use with RadPC and adapting its techniques to these other PCS controllers. Currently, a condensed version of this thesis has been accepted to the 2025 IEEE Intermountain Engineering, Technology, and Computing (i-ETC) Conference, to be presented and published in the conference proceedings [23].

FUTURE WORK

Future improvements on this body of research could include the development of functionalities for the verification tool to verify firmware uploaded to the board. For the RadPC, this may involve soldering a physical tap to the board's Rx and Tx lines, allowing the lines to be tapped without predefined ports to transmit the forwarded signals of the lines. Similarly to the verification process for logic code, any incoming bitstreams would be compared to a known good bitstream using SHA-256 hashing. Additionally, for future versions, any firmware the program deems trustworthy may be allowed to send through, rather than having one set trustworthy firmware. This may be implemented in part by inclusion of the Yara library, which provides binary file analysis to allow the verification tool to determine the trustworthiness of firmware binary files [1]. Lastly, the mitigative functionality of this tool may be applied to verification tools for PLCs and other applicable PCS controllers.

REFERENCES CITED

[1] YARA - The pattern matching swiss knife for malware researchers — virustotal.github.io. https://virustotal.github.io/yara/. [Accessed 20-11-2024].

[2] Wael Alsabbagh and Peter Langendörfer. A flashback on control logic injection attacks against programmable logic controllers. *Automation*, 3(4):596–621, 2022.

[3] Muhammad Rizwan Asghar, Qinwen Hu, and Sherali Zeadally. Cybersecurity in industrial control systems: Issues, technologies, and challenges. *Computer Networks*, 165:106946, 2019.

[4] Cataldo Basile, Stefano Di Carlo, and Alberto Scionti. Fpga-based remote-code integrity verification of programs in distributed embedded systems, 2012.

[5] Kejsi Biçoku. Security vulnerabilities in industrial plc software: A taxonomy and a systematic mapping study. 2021.

[6] Eric Byres and Justin Lowe. The myths and facts behind cyber security risks for industrial control systems. In *Proceedings of the VDE Kongress*, volume 116, pages 213–218, 2004.

[7] Jose Eduardo Urrea Cabus, İsmail Bütün, and Robert Lagerström. Security considerations for remote terminal units. In *2022 IEEE Zooming Innovation in Consumer Technologies Conference (ZINC)*, pages 47–52. IEEE, 2022.

[8] D.T. Campbell, J.C. Stanley, and N. L. Gage. *Experimental and Quasi-Experimental Designs for Research*. Houghton Mifflin, Boston, MA, 1963.

[9] Alvaro A. Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ASIACCS '11, page 355–366, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450305648. doi: 10.1145/1966913.1966959. URL https://doi.org/10.1145/1966913.1966959.

[10] Hao Chen and Luyang Xu. Software architecture and framework for programmable automation controller: A systematic literature review and a case study, 2018.

[11] FTDI Chip. C232hd usb 2.0 hi-speed to uart cable datasheet. https://ftdichip.com/wp-content/uploads/2023/05/DS_C232HD_UART_CABLE.pdf, (n.d.).

[12] Resilient Computing. Products. https://resilient-computing.com/products/, .

[13] Resilient Computing. Risc-v gnu compiler toolchain. https://github.com/riscv-collab/riscv-gnu-toolchain, .

[14] Thomas D Cook and D T Campbell. *Quasi-Experimentation: Design and Analysis Issues for Field Settings*. Houghton Mifflin, 1979.

[15] The MITRE Corporation. Common weakness enumeration. https://cwe.mitre.org/, (2024).

[16] The MITRE Corporation. Mitre att&ck ics matrix. https://attack.mitre.org/matrices/ics/, (n.d.).

[17] Dawoud Shenouda Dawoud and Peter Dawoud. *Serial communication protocols and standards*. River Publishers, 2022.

[18] Wei Gao and Thomas Morris. On cyber attacks and signature based intrusion detection for modbus based industrial control systems, 2014.

[19] Gigamon. Understanding network taps. https://www.gigamon.com/resources/resource-library/white-paper/understanding-network-taps-first-step-to-visibility.html, 2023.

[20] Emma Good. Security analysis of a siemens sicam cmic remote terminal unit, 2020.

[21] Arthur Grisel-Davy and Sebastian Fischmeister. Independent boot process verification using side-channel power analysis. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, pages 196–207, 2023. doi: 10.1109/QRS-C60940.2023.00053.

[22] Cybersecurity & infrastructure Security Agency. Irgc-affiliated cyber actors exploit plcs in multiple sectors, including us water and wastewater systems facilities. https://www.cisa.gov/news-events/cybersecurity-advisories/aa23-335a, Dec 2024. Accessed: 04/10/25.

[23] Christine Johnson, Ann Marie Reinhold, Clemente Izurieta, Bradley M. Whitaker, and Brock J. Lameres. Verification tool for securing risc-v fpga-based process control system. In *2025 Intermountain Engineering, Technology and Computing (IETC)*, page TBD. IEEE, 2025.

[24] Jaewon Kim, Woo-Hyun Ko, and P.R. Kumar. Cyber-security with dynamic watermarking for process control systems. In *2019 AIChE Annual Meeting*, 2019. URL https://www.researchgate.net/profile/Jaewon-Kim-5/publication/364657063_Cyber-security_with_dynamic_watermarking_for_process_control_systems/links/6356146f8d4484154a2b4c46/Cyber-security-with-dynamic-watermarking-for-process-control-systems.pdf.

[25] Johannes Klick, Stephan Lau, Daniel Marzin, Jan-Ole Malchow, and Volker Roth. Internet-facing plcs-a new back orifice. *Blackhat USA*, pages 22–26, 2015.

[26] Neha R Laddha and AP Thakare. A review on serial communication by uart. *International journal of advanced research in computer science and software engineering*, 3(1), 2013.

[27] Lucille McMinn and Jonathan Butts. A firmware verification tool for programmable logic controllers. In *Critical Infrastructure Protection VI: 6th IFIP WG 11.10 International Conference, ICCIP 2012, Washington, DC, USA, March 19-21, 2012, Revised Selected Papers 6*, pages 59–69. Springer, 2012.

[28] Mohamed Mesbah, Mahmoud Said Elsayed, Anca Delia Jurcut, and Marianne Azer. Analysis of ics and scada systems attacks using honeypots. *Future Internet*, 15(7), 2023. doi: https://doi.org/10.3390/fi15070241. URL `https://www.mdpi.com/1999-5903/15/7/241`.

[29] Eric Monmasson and Marcian N. Cirstea. Fpga design methodology for industrial control systems—a review. *IEEE Transactions on Industrial Electronics*, 54(4):1824–1842, 2007. doi: 10.1109/TIE.2007.898281.

[30] Eric Monmasson and Marcian N Cirstea. Fpga design methodology for industrial control systems—a review. *IEEE transactions on industrial electronics*, 54(4):1824–1842, 2007.

[31] Eric Monmasson, Lahoucine Idkhajine, Marcian N Cirstea, Imene Bahri, Alin Tisan, and Mohamed Wissem Naouar. Fpgas in industrial control applications. *IEEE Transactions on Industrial informatics*, 7(2):224–243, 2011.

[32] U.S. Department of Homeland Secutrity. Rechargeable batteries technote. `https://www.dhs.gov/sites/default/files/publications/RechargeableBatteries_TN_0508-508.pdf`.

[33] Jeff Payne. How to choose the best controller for each application: Consider the features and typical applications for industrial controllers to understand what type of controller fits each application. do you need a programmable logic controller (PLC), programmable automation controller (PAC), or industrial PC (IPC)? see comparison tables for PLCs, PACs, and IPCs. *28+. Gale Academic OneFile*, 64, 2017.

[34] Sasanka Potluri and Sasanka Diedrich. Deep learning based efficient anomaly detection for securing process control systems against injection attacks. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, 2019. URL `https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8843140`.

[35] Bradley Reaves and Thomas Morris. Discovery. infiltration, and denial of service in a process control system wireless network, 2009.

[36] Lucas L Ritzdorf, Colter Barney, Christopher M Major, Tristan Running Crane, Hezekiah Austin, Benjamin Macht, Clemente Izurieta, and Brock J LaMeres. Evaluating the effectiveness of obfuscated instruction codes for malware resistance. In *2023 Intermountain Engineering, Technology and Computing (IETC)*, pages 67–72, 2023. doi: 10.1109/IETC57902.2023.10152036.

[37] Movable Type Scripts. Sha-256 cryptographic hash algorithm. https://www.movable-type.co.uk/scripts/sha256.html, (n.d.).

[38] Shreyas Sharma. Risc-v architecture: A comprehensive guide to the open-source isa. https://www.wevolver.com/article/risc-v-architecture-a-comprehensive-guide-to-the-open-source-isa, 2023.

[39] K Stouffer, V Pillitteri, S Lightman, et al. Guide to industrial control systems security (nist sp 80082). *NIST Special Publication 800-82*, 2014.

[40] MIT Electric Vehicle Team. A guide to understanding battery specifications, Dec 2008. URL https://web.mit.edu/evt/summary_battery_specifications.pdf.

[41] Zibo Wang, Yaofang Zhang, Yilu Chen, Hongri Liu, Bailing Wang, and Chonghua Wang. A survey on programmable logic controller vulnerabilities, attacks, detections, and forensics. *Processes*, 11(3), 2023. ISSN 2227-9717. doi: 10.3390/pr11030918. URL https://www.mdpi.com/2227-9717/11/3/918.

[42] Haroon Wardak, Sami Zhioua, and Ahmad Almulhem. Plc access control: a security analysis. In *2016 World Congress on Industrial Control Systems Security (WCICSS)*, pages 1–6, 2016. doi: 10.1109/WCICSS.2016.7882935.

[43] Joe Weiss. Iran hacks us water system: Observation and implications of a terrorist attack on us soil. https://www.controlglobal.com/blogs/unfettered/blog/33015578/iran-hacks-us-water-system-observation-and-implications-of-a-terrorist-attack-on-us Nov 2023. Accessed: 04/10/25.

[44] Haolan Wu, Yangyang Geng, Ke Liu, and Wenwen Liu. Research on programmable logic controller security. In *IOP Conference Series: Materials Science and Engineering*, volume 569, page 042031. IOP Publishing, 2019.

[45] Hyunguk Yoo and Irfan Ahmed. Control logic injection attacks on industrial control systems. In *IFIP International Information Security Conference*, 2019. URL https://api.semanticscholar.org/CorpusID:189926624.

[46] Hyunguk Yoo, Sushma Kalle, Jared Smith, and Irfan Ahmed. Overshadow plc to detect remote control-logic injection attacks. In Roberto Perdisci, Clémentine Maurice, Giorgio Giacinto, and Magnus Almgren, editors, *Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 109–132, Cham, 2019. Springer International Publishing. ISBN 978-3-030-22038-9.

APPENDIX

STATISTICAL ANALYSIS CODE

```
% R programming language code used to perform multiple linear
    regression analysis and simple linear regression on PCS performance
    data.
library(tidyverse)
library(gtsummary)
library(survival)
library(readxl)

# Read the Excel file (adjust path as needed)
df <- read.csv("D:/Documents/ICS Research/entire_dataset_mlr.csv")

# Redefine data column names
colnames(df)[1:4] <- c("BatVol", "HasTool", "Time", "CLTVol")

# Fit the multiple linear regression model
entire_model <- lm(BatVol ~ HasTool + Time + CLTVol, data = df)

# View the regression output
summary(entire_model)

tool_model <- lm(BatVol ~ HasTool, data = df)
summary(tool_model)

time_model <- lm(BatVol ~ Time, data = df)
summary(time_model)

ctl_model <- lm(BatVol ~ CLTVol, data = df)
summary(ctl_model)
```