Chapter 9

CYBER-RESILIENT SCADA SYSTEMS VIA SECURE STATE RESTORATION

Zachary Birnbaum, Matthew Davis, Salman Salman, James Cervini, Lanier Watkins, Saikiran Yamajala and Shruti Paul

Abstract Supervisory control and data acquisition (SCADA) systems are widely used in the critical infrastructure. These systems are high risk targets for cyber attacks due to their criticality, interconnectedness and Internet accessibility. SCADA systems employ programmable logic controllers to monitor and issue control instructions to other devices. Unfortunately, programmable logic controllers are typically configured in a persistent manner – they are configured once and designed to operate continuously. They are, therefore, ill-suited to operate in virtual, dynamic and cyber-resilient environments. SCADA systems must employ cyber-resilient architectures to enable them to endure and recover from cyber attacks.

This chapter describes a secure methodology for storing SCADA system states that can be used by redundant, non-persistent devices during operations and recovery. The proposed methodology realizes a non-persistent, Byzantine fault-tolerant, virtual industrial control system architecture whose state and function can be stored and restored securely, contributing to its cyber resilience. Implementation of the methodology in a SCADA environment incorporating non-persistent programmable logic controllers reveals that cyber attacks are identified quickly and secure restoration can occur without loss of state or functionality. Mathematical and timing analyses demonstrate the applicability and efficacy of the methodology in creating cyber-resilient SCADA systems.

Keywords: SCADA systems, non-persistence, fault tolerance, cyber resilience

1. Introduction

Supervisory control and data acquisition (SCADA) systems are essential to monitoring and controlling industrial operations across the critical infrastructure. Unlike many enterprise computing systems, these com-

The original version of this chapter was revised: the name of the fourth author was changed. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-62840-6_17

J. Staggs and S. Shenoi (Eds.): Critical Infrastructure Protection XIV, IFIP AICT 596, pp. 183–207, 2020.
https://doi.org/10.1007/978-3-030-62840-6_9

plex cyber-physical systems have tight real-time constraints due to their interactions with the physical world.

The critical and interconnected nature of SCADA systems makes them attractive targets for cyber attacks. The Stuxnet attacks [19] demonstrated the vulnerabilities of SCADA systems and the irreparable damage that can be caused. Stuxnet, a self-replicating computer virus, infected programmable logic controllers (PLCs) used in Iran's nuclear program, destroying more than 1,000 uranium enrichment centrifuges and significantly impacting Iranian nuclear ambitions.

Securing SCADA systems from cyber attacks is challenging. Classical defensive techniques such as encryption, firewalls, anomaly detection and patching are difficult to apply in SCADA environments. Something as simple as applying a system update, which occurs with regularity in enterprise environments, is challenging due to the high availability demands of SCADA systems. Additionally, many SCADA systems were designed and integrated years ago and, therefore, have limited computing, memory and networking resources that hinder the ability to implement modern security mechanisms.

Applying conventional cyber defense and attack prevention solutions such as encryption, authentication and anomaly detection to SCADA systems may not be enough to deter skilled and determined adversaries. Therefore, SCADA systems must rapidly recover their functionality after they are degraded or disrupted. Cyber resilience is the ability to continuously deliver the intended outcomes despite adverse cyber events – it goes beyond attack prevention by ensuring continuity of operations. A National Institute of Standards and Technology special publication [24] lists 14 resilience techniques, including diversity, unpredictability, non-persistence and redundancy.

Virtualization is a promising technology for implementing defensive and cyber resilience techniques in SCADA environments. Many vendors and integrators now offer virtual SCADA and industrial control system components. In fact, during new integrations and upgrades, older SCADA components are increasingly being replaced by their virtual counterparts. Due to the increased computing power, memory and network bandwidth, virtualization enables security techniques to be applied in SCADA environments. Unlike physical systems, virtual systems can be rebooted on demand. Rebooting a system, a simple cyber-resilient (and defensive) action that manifests non-persistence, is effective against certain types of malware [15].

Unfortunately, applying cyber resilience techniques in SCADA environments is challenging, primarily due to their real-time nature, high availability requirements and criticality. For example, when manifest-

ing non-persistence via a system reboot, it is important to minimize the downtime. Applying non-persistence techniques to SCADA systems is also challenging; rapidly regaining positive control is vital, but positive control often requires knowledge of the current inputs as well as past inputs and outputs. The reliance on historical data directly conflicts with non-persistence. Other resilience techniques, such as redundancy and heterogeneity, are easily integrated in cyber-resilient solutions for SCADA systems without this concern.

Novel solutions must be developed to imbue cyber resilience in SCADA systems. This chapter describes a secure methodology for storing system states that can be used by redundant, non-persistent devices during operation and recovery. The methodology realizes a non-persistent, Byzantine fault-tolerant, virtual industrial control system architecture whose state and function can be stored and restored securely, contributing to its cyber resilience.

2. Background

This section briefly discusses important aspects of control theory, SCADA system virtualization and cyber resilience.

2.1 Control Theory

The primary objective of a SCADA system is to produce outputs (control commands to actuators) in the face of input disturbances [26]. Outputs are computed via feedforward or feedback mechanisms. Feedforward control reduces the effects of measured input disturbances by adjusting the control effort based on the input disturbances [21]. Feedback control measures the difference between the true and desired system states and uses the error to adjust the control effort [2]. This section considers feedback systems, but a similar treatment can be applied to feedforward systems.

Control systems are typically designed in the Laplace domain that transforms hard-to-solve continuous time-domain differential equations to their simple frequency domain equivalents [26]. The designed solutions are then converted to discrete time difference equations for implementation in modern controllers.

For example, a third-order continuous time-domain controller can be represented in the Laplace domain $\mathcal{L}$ as:

$$H(s) = \frac{a_2 s^2 + a_1 s + a_0}{b_3 s^3 + b_2 s^2 + b_1 s + b_0} \tag{1}$$

where a_i and b_i are constants, and s is a complex number frequency parameter.

Assuming a constant time step Δt and adjusting for causality, the equivalent representation in the discrete time domain $\mathcal{Z}$ is:

$$H(z) = \frac{\alpha_2 z^{-1} + \alpha_1 z^{-2} + \alpha_0 z^{-3}}{1 + \beta_2 z^{-1} + \beta_1 z^{-2} + \beta_0 z^{-3}} \tag{2}$$

where α_i and β_i are constants, and z is the discrete complex time parameter.

The corresponding $\mathcal{Z}$ domain controller is represented by the following difference equation:

$$\begin{aligned} y(k) = & -\beta_2 y(k-1) + -\beta_1 y(k-2) + -\beta_0 y(k-2) + \\ & \alpha_2 x(k) + \alpha_1 x(k-1) + \alpha_0 x(k-2) \end{aligned}$$

where α_i and β_i are constants, and $y(k)$ is the output at the discrete time step k.

This difference equation is easily implemented by a controller that transforms inputs to outputs consistent with the system design. Even in this fairly simple control system example, previous input and output states are key to computing the new output. The controller can then be integrated in a standalone physical control device or virtualized in a modern industrial control system architecture.

2.2 SCADA System Virtualization

The concept of a virtual SCADA environment is still novel and its utility is actively discussed in the industrial community. In a virtual SCADA system deployment, a single hypervisor typically contains multiple virtual programmable logic controller instances.

Johansson [18] discusses the benefits and disadvantages of SCADA system virtualization. The benefits include:

- **Number of Physical Devices:** Co-locating multiple virtual programmable logic controllers and human-machine interfaces reduces the physical footprints of SCADA systems.

- **Software Development and Disaster Recovery:** Virtual machine snapshots facilitate quick system rollbacks to saved copies after failures due to software errors, cyber attacks and catastrophes.

- **Security Architecture Testing:** Virtual security solutions are readily deployed in SCADA environments. Security can be inte-

grated in hypervisors and standard host-based solutions can be incorporated directly in virtual programmable logic controllers.

- **System Updates:** Virtual systems are easier to update and patch compared with their physical counterparts. A patch can be tested and integrated in a virtual environment upon verification of functionality and the system can be rolled back to a saved image in the event of a failure.

The disadvantages of SCADA system virtualization include:

- **System Complexity:** Adding new technology to a system makes it more difficult to identify the origin of a system problem or failure.

- **Non-Standard Physical Interfaces:** Many SCADA environments use proprietary or outdated physical devices to interface with sensors and actuators. Interfacing these devices with commodity server equipment is challenging.

- **Cyber Attack Surface:** Hypervisors increase the attack surface. Successfully compromising a hypervisor may provide an attacker with root privileges to all the running virtual machines [4]. Developing cyber secure hypervisors with reduced attack surfaces is an active area of research [27].

The proposed methodology is designed for virtual SCADA systems. The benefits of virtualization outweigh the disadvantages, which can be mitigated by technical maturity and careful adoption [25]. Additionally, advancements have been made in virtual SCADA devices and testbeds [22, 23]. The application of virtualization to SCADA systems enhances cost savings and the ease of integration. This chapter demonstrates that SCADA system virtualization increases cyber resilience.

2.3 Cyber Resilience

Cyber resilience is a relatively new concept in the cyber security field. It involves several key characteristics [6]:

- **Inhibit:** While inhibit capabilities are technically not an official characteristic of cyber resilience, they prevent an adversary from having a cyber impact on a system. Traditional security techniques are often employed to enhance the overall cyber resilience.

- **Endure:** Endure capabilities enable a system to provide a minimum level of functionality when it is under cyber attack.

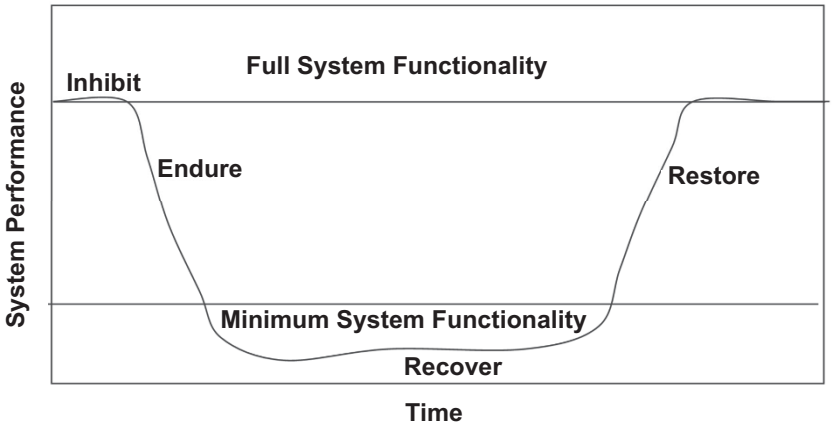

Figure 1. Cyber resilience capabilities timeline.

- **Recover:** Recover capabilities enable actions to be undertaken to respond to degraded system performance.
- **Restore:** Restore capabilities help regain complete system functionality.
- **Improve:** Improve capabilities support enhanced cyber resilience design and implementation to mitigate anticipated threats.

Figure 1 shows the placement of the five cyber resilience characteristics in an operational timeline. Inhibit technologies keep a system fully functional. However, if a cyber effect is realized despite the defenses, endure capabilities ensure that the system can still provide minimum functionality. Recover capabilities ensure that, even if the system functionality falls below an acceptable threshold, adequate functionality will become available in an acceptable timeframe. Finally, restore capabilities ensure that the system will regain full functionality.

Cyber resilience capabilities are realized through the application of several techniques. Bodeau and Graubart [6] have identified the following cyber resilience techniques:

- **Applied Analytics and Monitoring:** This refers to the continual collection and analysis of system data to identify possible vulnerabilities, adversarial activities and negative system impacts.
- **Heterogeneity:** This refers to the utilization of diverse cyber technologies to minimize the impacts of attacks and force an adversary to attack different technologies with different vulnerabilities.

- **Distributed Allocation:** This refers to the positioning of critical assets to provide an unpredictable attack surface to an adversary.
- **Non-Persistence:** This refers to the retention of systems for a limited time, reducing the ability of an adversary to act maliciously and establish a persistent foothold. A persistent system is generally configured once and accessed only if troubleshooting or maintenance is required. A non-persistent system is designed to support repeated shutdown, destruction, re-creation and initialization.
- **Redundancy:** This refers to the presence of multiple protected instantiations of a system, forcing an adversary to achieve cyber effects on multiple targets simultaneously.

3. Related Work

Cardenas et al. [7] have provided an overview of the challenges involved in securing SCADA systems. Many challenges arise from the unique properties of SCADA systems compared with enterprise environments, namely their legacy nature and real-time operational constraints.

Melin et al. [20] have demonstrated that traditional cyber security techniques can be applied to SCADA systems, but the systems must be designed to be cyber resilient. Their work focuses on a framework and testing criteria for system resilience. A key finding is that resilience is greatly increased by eliminating the ability to remotely program individual programmable logic controllers. This is because an attacker is limited to adjusting only the system reference, which is easily detected.

Cox et al. [10] have demonstrated that heterogeneity provides cyber resilience. Their framework executes heterogeneous system variants with the same inputs and monitors their behavior to detect anomalies; this technique forces an attacker to compromise multiple system variants to achieve the desired effects. Gearheart et al. [14] have specified diversity metrics and have demonstrated that text-based features can effectively differentiate software diversity strategies implemented in open-source diversifying compilers.

Byzantine fault-tolerant systems, which employ a form of distributed allocation or redundancy, are also effective at enhancing cyber resilience. Considerable work has been done in this area, especially coupled with virtualization and other resilience enabling technologies.

Ahmed and Bhargava [1] have demonstrated that Byzantine fault avoidance can be implemented in cloud environments using OpenStack and software defined networking by allowing replicas to live for a short time on computing platforms (hypervisors, hardware and operating systems) as a form of moving target defense. They present a fault avoidance

architecture that leverages cloud platform technologies, providing replica refresh algorithms to control the exposure of platforms to attacks, and a scheme that preserves state while undergoing failure avoidance.

A Byzantine architecture employs a set of replicas for failure avoidance. Within the replicas, a primary node exchanges consensus messages with a specific replica. The replica node then prepares a reply and commits the response. The node is selected from a pre-prepared virtual machine pool, which is refreshed after a set amount of time or after completing a specific number of transactions. The primary benefit is that the time advantage to a potential attacker is eliminated by not having a virtual machine operate indefinitely. By dynamically using a cloud software stack, Ahmed and Bhargava [1] were able to minimize the virtual machine attack exposure window and boot new virtual machines in under 12 seconds. Unfortunately, while the approach has its benefits, a 12-second recovery time is often unacceptable in real-time SCADA environments.

Babay et al. [3] have created an intrusion-tolerant SCADA system that is resilient to system-level compromises and network attacks. Their approach uses the Spines messaging framework, which provides automatic reconfiguration and network flexibility [12]. Core SCADA functionality is distributed across $3f + 2k + 1$ heterogeneously-compiled (multicompiler) replicas, where f is the number of simultaneous acceptable failures and k is the number of proactive recoveries (with no failures detected). The replicas themselves are distributed across multiple cloud servers. The SCADA system was employed in a power grid experiment, where it met all the power and latency requirements. However, Babay and colleagues do not address how programmable logic controller failures are detected and how the system state can be transitioned correctly to new environments.

Yamamoto et al. [29] have employed a customized intrusion detection solution and virtualization technologies to enable SCADA system recovery. Running multiple virtual programmable logic controllers simultaneously supports seamless transition from an anomalous virtual programmable logic controller to another virtual controller. Regions of acceptable behavior are defined to enable anomaly detection and trigger a transition to the backup virtual programmable logic controller. However, Yamamoto and colleagues do not address the inherent issues associated with homogeneity and cyber attacks – if the primary and backup systems are homogeneous (because they use the same baseline virtual image), an attack launched against the primary system would also impact the backup system. Additionally, Yamamoto et al. do not

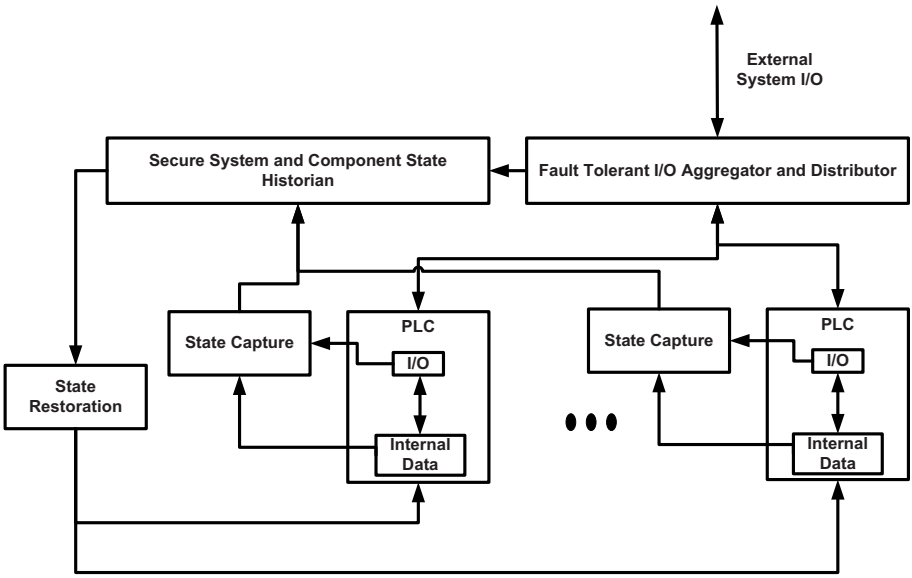

Figure 2. Proposed cyber-resilient architecture.

address state restoration directly; they simply assume it will be maintained by the failover system.

Cyber-resilience techniques have been developed for enterprise and SCADA system applications in standard and virtual environments. However, the proposed cyber resilience methodology is unique in that it focuses on SCADA system endurance and restoration in non-persistent, redundant virtual environments.

4. Proposed Methodology

This section describes the methodology for achieving cyber resilience in SCADA systems using state restoration.

4.1 Overview

Rapid and reliable state restoration are important requirements for cyber-resilient SCADA system architectures with non-persistence. Employing redundancy further increases cyber resilience. The proposed cyber resilience methodology involves state storage and restoration in a non-persistent, redundant SCADA environment.

Figure 2 shows the key features of the proposed cyber-resilient architecture. These include:

- **State Capture:** The system state is identified and captured without negatively impacting performance.
- **State Storage:** After the state has been captured, it is stored in a secure, immutable and decentralized manner.
- **State Recovery:** When a new, non-persistent SCADA device is added to the system, the state is pushed promptly to the new device to ensure continued operation.
- **I/O Aggregation and Distribution:** The inputs and outputs to and from the redundant, non-persistent SCADA components are processed securely.
- **Non-Persistence:** Non-persistent redundant programmable logic controllers and other devices are started, stopped and restarted with relatively little effort.

Implementing these features requires a complex and capable architecture. The proposed architecture enables continuity of normal operations due to component redundancy and ensures rapid state recovery of SCADA components when required. To simplify the presentation, but without any loss of generality, it is assumed that programmable logic controllers are the only SCADA devices that have non-persistent and redundant capabilities.

The new architecture functions as follows during normal operations:

1. Input is sent to the I/O aggregator and distributor via normal communications paths. Communications in a SCADA environment may use a point-to-point serial protocol, Ethernet or wireless.
2. The I/O distributor sends inputs to multiple programmable logic controllers for processing. Ideally, these programmable logic controllers would be heterogeneous, further enhancing cyber resilience.
3. After a programmable logic controller completes its scan and computes its output, the internal programmable logic controller data structures and memory are saved to a state storage database (blockchain) [28].
4. The I/O aggregator processes the output from each programmable logic controller using a Byzantine fault-tolerant algorithm to determine consensus [3].
5. The I/O aggregator stores the input data, consensus and associated metadata in the blockchain.

6. The resulting consensus state is converted to an output, which the I/O aggregator transmits to an actuator to implement a control action.

A state restoration procedure is executed when any of the following conditions hold:

- An intrusion or cyber attack is detected on a non-persistent programmable logic controller.
- A programmable logic controller does not agree with the output of the consensus voting scheme.
- A programmable logic controller is randomly selected to be restarted based on another factor (e.g., random sampling, time-alive or pre-determined schedule). In order to prevent an adversary from obtaining access to all the programmable logic controllers prior to launching an attack (which would not be detected by the fault-tolerant voting scheme), the virtual programmable logic controllers should be destroyed and restored at irregular, but operationally-small, time intervals to ensure non-persistence.

The state restoration procedure, which is executed on every programmable logic controller, has the following steps:

1. The virtual programmable logic controller is destroyed, a trivial task in a virtual environment.
2. A new programmable logic controller is initialized to a default state containing no internal data. Ideally. the new programmable logic controller should be heterogeneous compared with the previous version, limiting the likelihood that a previous cyber attack would be successful.
3. The last known good state of the system is identified using the state database and the corresponding programmable logic controller internal memory and data are pushed by the state restoration engine to the new programmable logic controller.

The remainder of this section describes how the system state is captured, determined, stored and restored in a secure manner.

4.2 Capturing State

Before capturing the complete SCADA system state, it is necessary to identify the locations where the data resides. Figure 3 shows the

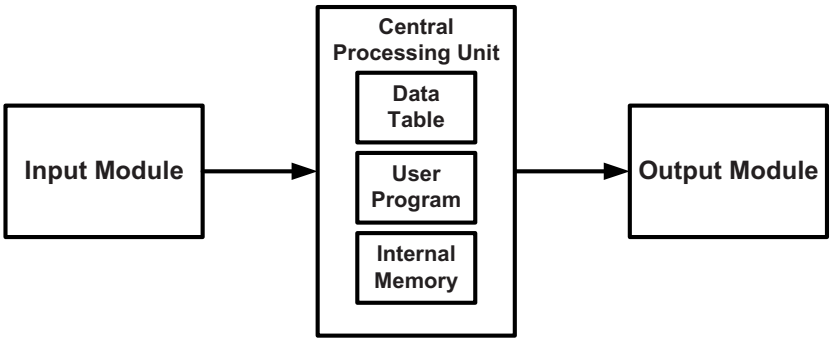

Figure 3. Programmable logic controller components.

components of a standard programmable logic controller. In order to capture the system state, it is necessary to collect input data, output data and data residing in internal memory structures.

Since modern SCADA systems employ enterprise components for communications [16], the costs of capturing input and output information are minimal. If standard Ethernet is used, a physical network tap or modified switch can be used to monitor data going to and from devices. If the device is virtual, equivalent software solutions are available to capture network data. Regardless of the approach used, it is important that state capture does not negatively impact SCADA system operations.

The process for capturing internal programmable logic controller data varies according to the vendor hardware. For example, if the Modbus communications protocol is used and internal data is mapped to holding registers, it is straightforward to request internal programmable logic controller data over a network connection [9].

Programmable logic controllers have very predictable scan cycles, with each cycle taking several milliseconds. Each scan cycle has input, program and output stages [28]:

- **Input Stage:** Input is read by the programmable logic controller CPU from the input modules.
- **Program Stage:** Input and current internal data and memory structures are modified as necessary.
- **Output Stage:** Required changes to the output modules are made.

The state capture must operate after every scan cycle without degrading system performance.

4.3 Determining State

After the I/O and internal state of each device have been captured, the overall system state is computed for each device and anomalous states and devices are identified. The Byzantine fault tolerance algorithm of Castro and Liskov [8] with $3f + 1$ replicas is employed to determine consensus (f is the maximum number of possible faulty nodes).

The following procedure computes the consensus state and identifies anomalous devices for destruction and restoration:

1. Collect the output information state from $3f + 1$ devices.
2. Determine the consensus state as the state common to at least $f + 1$ devices.
3. Identify the non-consensus devices for restoration.
4. Save the consensus state in the blockchain.

4.4 Storing State

After the system state has been determined via secure means, it must be stored in a secure manner. Specifically, it is necessary to ensure that the stored system state can be recovered at any future point in time with the assurance that it was not modified.

A blockchain can be used to ensure non-repudiation and immutability [11]. Each iteration involving state capture, consensus determination and metadata addition results in a data block. Decentralized operation prevents an attacker from modifying the shared history from a single location, which enhances security. Different blockchains are used for the I/O aggregator and distributor, and for each programmable logic controller. If the programmable logic controllers are heterogeneous, then a separate blockchain must be maintained for each programmable logic controller. If the programmable logic controllers are homogeneous, then all the internal data structures and memory are identical, and a single blockchain may be used.

Each I/O aggregator data block contains the following information: (i) input value; (ii) consensus state; (iii) time; (iv) iteration number; (v) hash value of the previous block; (vi) hash value of all the current block data; (vii) programming logic controllers that agree with the consensus; and (viii) programming logic controllers that disagree with the consensus (i.e., devices that have to undergo state restoration).

Each programmable logic controller data block contains the following information: (i) input value; (ii) output value; (iii) internal data and

memory; (iv) time; (v) iteration number; (vi) hash value of the previous block; and (vii) hash value of all the current block data.

After the block information has been computed, the block is appended to the corresponding blockchain that contains all the preceding blocks. At this point, the system state is securely stored in an immutable manner and is ready to be restored when required.

4.5 Restoring State

After each data block is appended to the I/O aggregator chain, the virtual programmable logic controllers that were identified as having disagreed with the consensus are destroyed. Such destruction is a routine and timely procedure in virtual environments.

A new virtual programmable logic controller is then initialized. However, before it can process any input, the blank internal structures are updated with historical consensus data. Note that the restored state is independent of the specific programmable logic controller instance; instead, it represents the consensus of all the participating programmable logic controllers.

The following steps are involved in initializing a new programmable logic controller:

1. Identify the last several consensus states required for successful programmable logic operation.

2. For each identified state, select a random programmable logic controller with an output equal to the consensus state.

3. For each identified state, pull the appropriate internal information from the blockchain of the selected programmable logic controller and save the information in the internal data structures of the new programmable logic controller.

These steps assume that the programmable logic controllers are homogenous. Depending on the programmable logic controller vendor, virtualization technology and other factors, writing to the internal programmable logic controller structures has a varying degree of difficulty. If the Modbus protocol is used, it is possible to write directly to the programmable logic controller data structures via normal communications protocols; this task can be more challenging for other programmable logic controllers.

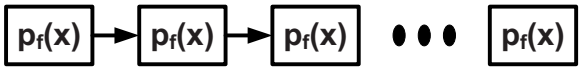

Figure 4. Iteration flow of the architectures.

5. Experimental Verification

The experimental verification involved two types of analyses. The first was a mathematical analysis to verify that the proposed methodology increases cyber resilience. The second involved a timing analysis to determine the overhead induced by the architecture.

5.1 Mathematical Analysis

Three SCADA system architectures were analyzed to assess the efficacy of the proposed methodology:

- Single standard programmable logic controller without redundancy and state restoration.
- Byzantine fault-tolerant architecture without state restoration.
- Byzantine fault-tolerant architecture with state restoration.

Each architecture was analyzed as a process over time to determine when a failure threshold of $p_t(x)$ was reached. Each iteration of the architecture assessed the ability to produce a correct output while assuming a constant failure probability of $p_f(x)$. Assuming that the probability of architecture failure $p_f(x)$ was greater than zero meant that, at some point in time t or after a certain number of iterations, the overall process failure probability would be greater than the failure threshold $p_t(x)$.

Figure 4 shows the iteration flow of the architectures. As an architecture becomes more resilient, the probability of an iteration being successful increases, which increases the expected longevity of the architecture. The overall process can be analyzed using the following equation:

$$p_t(x) < 1 - (1 - p_f(x))^i \tag{3}$$

where i is the number of expected iterations until system failure.

In the case of the Byzantine architectures, at least $f + 1$ of the $3f + 1$ individual programmable logic controller operations had to succeed in order for an overall iteration to be successful. Therefore, the following

Table 1. Expected iterations to failure for various architectures ($p_t(x) = 0.99$).

Architecture	Failure Probability	Redundant PLCs	Expected Iterations to Failure
Single PLC	0.1	0	44
Single PLC	0.2	0	21
Single PLC	0.4	0	10
Byzantine	0.1	1	1,243
Byzantine	0.2	1	167
Byzantine	0.4	1	24
Byzantine	0.1	3	504,862
Byzantine	0.2	3	5,326
Byzantine	0.4	3	82
Byzantine	0.1	5	170,381,911
Byzantine	0.2	5	141,198
Byzantine	0.4	5	239

modified computation of $p_f(x)$ was performed:

$$p(\text{at least } m \text{ out of } n) = \sum_{i=m}^{n} \binom{n}{i} p^i (1-p)^{n-i} \tag{4}$$

where f is the number of tolerable failures, $m = f + 1$, $n = 3f + 1$, $p = 1 - p_f(x)$ and i is the number of expected iterations until system failure.

Two variables were analyzed while also varying the architectures: (i) probability of architecture failure $p_f(x)$ during a single iteration; and (ii) number of redundant programmable logic controllers in a Byzantine fault-tolerant architecture.

Table 1 shows the results. As the probability of an individual iteration failure increased, the expected system longevity decreased. Furthermore, the Byzantine architectures without state restoration were more resilient than the single programmable logic controller architectures without redundancy and state restoration.

It is important to note that the expected iterations until failure for the Byzantine architectures with state restoration could not be computed. This is because every programmable logic controller failure was corrected after an interaction and the architecture was restored to the correct state before it received any subsequent inputs. Therefore, each iteration was effectively independent and the overall probability of system failure as inputs were processed was never greater than $p_f(x)$ regardless of the

number of iterations or system uptime. Additionally, the overall system probability was less than $p_t(x)$.

However, the increased resilience using state restoration did not come without a cost. Additional computing resources and time were required to successfully identify and restore the anomalous programmable logic controllers.

5.2 Timing Analysis

A timing analysis of the architectures was conducted to assess the impact of the cyber resilience methodology. Three components were instantiated using Python programs: (i) programmable logic controllers; (ii) fault-tolerant I/O aggregator and distributor; and (iii) secure system and component state historian. The individual components communicated using the Flask micro web framework [17].

The environment did not adhere strictly to real-world SCADA system and programming logic controller restrictions, but it did incorporate the same building blocks used in real systems. The simulated programmable logic controllers processed difference equations much like their real-world counterparts. The other virtual components had no readily-available industry equivalents, so simulated Python representations were utilized.

Architecture Complexity. The complexity of the architecture was assessed using three variables:

- **Programming Logic Controller Complexity:** The length of the difference equation used by a programmable logic controller was used as a pseudo-approximation of its complexity.
- **System Redundancy:** The number of tolerable programmable logic controllers in a Byzantine fault-tolerant system was used to approximate system redundancy.
- **Blockchain State Storage Enabled:** A Boolean value indicated if the states of a programmable logic controller and I/O aggregator and distributor were committed to a blockchain.

The first timing analysis experiment examined the timing impacts of the programmable logic controller complexity variable. The other two variables were held constant. Table 2 shows the results – as the programmable logic controller complexity increased, so did the expected execution time.

The second timing analysis experiment assumed a Byzantine fault-tolerant architecture with a variable number of tolerable failed programmable logic controllers. All the other variables were held constant.

Table 2. Execution times of a single programmable logic controller.

PLC Coefficients	Execution Time (seconds)
1	0.011806
10	0.012572
100	0.013346
1,000	0.022429

The number of coefficients (i.e., programmable logic controller complexity) was set to 100 and blockchain state storage was not employed.

Table 3. Execution times of the I/O aggregator and distributor.

Tolerable PLC Failures	Execution Time (seconds)
0	0.020581
1	0.040104
2	0.061726
4	0.105565
8	0.186051

Table 3 shows that, as the number of nodes required for the Byzantine fault-tolerant system increased, so did the time required to process a single input. Also, a Byzantine fault-tolerant system with zero tolerable failures had an overhead of 54.2% compared with a single programmable logic controller of equal complexity.

The third timing analysis experiment examined the impact of state restoration. The single programmable logic controller and Byzantine fault-tolerant architectures were assessed with and without state saving after every scan cycle. In the experiment, the number of programmable logic controller coefficients was fixed at 100 and the number of tolerable failures was set to zero.

Table 4 shows that committing the system state to the blockchain after every iteration caused significant overhead. The overhead for a single programmable logic controller was 44.2% whereas the overhead was 88.8% for the Byzantine architecture. The increased overhead of the Byzantine architecture was caused by two commits to the blockchain, once for the programmable logic controller in the architecture and once after consensus was reached.

Table 4. Execution times of the secure state storage.

Experiment	Secure Storage (Blockchain) Used	Execution Time (seconds)
PLC	No	0.009254
PLC	Yes	0.013346
Byzantine	No	0.020581
Byzantine	Yes	0.038867

Restoration Analysis. The previous experiments assessed the architectural complexity, not resilience in the face of failures. This section discusses the experiments conducted to assess the timing impacts of restoration where the Byzantine fault-tolerant and state capture systems were enabled. When simulating an attack against a single programmable logic controller, it was assumed that the attack probability was constant and that each programmable logic controller iteration was equally likely to be attacked. Since the goal was not to determine when an architecture would fail, but to assess the timing impacts, it was assumed that only a maximum of f programmable logic controllers could fail where f is the number of tolerable programmable logic controller failures. Because at most only f programmable logic controllers failed, every architecture always returned the correct result.

Each architecture in the experiments had a predetermined probability of failure that was propagated to the programmable logic controllers. To determine if a programmable logic controller failed during each iteration, f programmable logic controllers generated a random value between zero and one. If the value was less than the probability of failure, the (anomalous) programmable logic controller yielded an incorrect result, forwarding the wrong answer to the I/O aggregator. After each iteration, the anomalous programmable logic controllers were restored.

Table 5 shows the results obtained when the number of tolerable programmable logic controller failures was varied while keeping the probability of a single failure constant at 40%. As expected, the greater the number of redundant devices, the greater the expected execution time. This was due to the increased number of queries that the I/O aggregator had to issue in order to achieve consensus and the increased number of commits to the blockchain.

Table 6 shows the results obtained when the probability of f programmable logic controllers failing was varied while holding all the other variables constant. The number of tolerable programmable logic controller failures was set to four and the number of programmable logic

Table 5. Execution times of the I/O aggregator and distributor with secure storage.

Tolerable PLC Failures	Execution Time (seconds)
1	0.076437
2	0.100038
4	0.165552
8	0.297060

Table 6. Execution times of the I/O aggregator and distributor with secure storage.

PLC Failure Rate	Execution Time (seconds)
0	0.151104
0.05	0.152491
0.1	0.153756
0.2	0.156992
0.4	0.165552
0.8	0.190867
1	0.194210

controller coefficients was 100. The table reveals that the expected execution time increased with the probability of failure, but the failure rate did not incur as much overhead as the other variables. These results indicate that the timing cost to restore a programmable logic controller was fairly low. Comparing the results for the architecture with 0% failure probability to the one with 100% failure probability reveals that restoring all f programmable logic controllers required only 28.5% overhead.

5.3 Summary

The mathematical analysis demonstrated that incorporating two redundant programmable logic controllers in a Byzantine fault-tolerant scheme along with state storage and restoration capabilities rendered each system iteration independent. This is important because programmable logic controllers use difference equations, so past computations are important for present and future computations. A key benefit of the cyber resilience methodology is that past failures cannot propagate to future operations.

The timing analysis demonstrated that implementing redundancy and blockchain state storage increased overhead. However, the overhead involved in restoring state to failed programmable logic controllers was comparatively fast.

6. Discussion

This section discusses the strengths and limitations of the proposed cyber resilience methodology.

6.1 Strengths

The primary strength of the methodology is that it realizes a cyber-resilient SCADA architecture that can mitigate the negative effects of a number of common cyber attacks. Due to non-persistence, redundancy and state restoration, an attack against a single programmable logic controller would not cause irreparable damage. The methodology supports the endurance and recovery phases of cyber resilience. Moreover, the architecture is simple to implement and could easily be retrofitted to existing SCADA systems. Additionally, the use of blockchain technology and fault-tolerant consensus ensures that the system state is computed and stored in a secure manner, which enhances cyber resilience.

Experimental evaluations of the methodology suggest that the implementation impact is fairly minimal given virtualization and modern computing resources, provided that the infrastructure exists in the SCADA environment. Most deployed industrial control systems have outdated physical hardware. As new systems are designed and integrated, transitions to modern virtual environments are likely to occur due to their reduced physical footprints, ease of upgrade and service co-location compared with physical environments. Thus, the proposed cyber resilience methodology is expected to see increased applications in future SCADA environments.

6.2 Limitations

The proposed methodology eliminates the reliance on single programmable logic controllers to enhance cyber resilience, but this increases the cyber attack surface. Instead of targeting a programmable logic controller, an attacker could target the consensus algorithm, blockchain and virtual programmable logic controller destruction and restoration procedures. Therefore, it is vital that all the additional components in a cyber-resilient SCADA architecture incorporate cyber security best practices to the extent possible. Certain improvements can be made to mitigate the increased cyber attack surface. For example, the blockchain

could be distributed and made non-persistent, ensuring that it could be restored from a distributed replica if it was attacked.

Creating virtual programmable logic controllers and other SCADA devices is by no means easy. Not all SCADA devices can be virtualized at this time and additional research is required. The proposed methodology was implemented and tested using standard enterprise equipment (i.e., laptops running Python code); the experiments did not engage industry-grade physical or virtual programmable logic controllers, let alone an industrial control system testbed. Further research is needed to evaluate the methodology in realistic SCADA environments. State restoration for virtual programmable logic controllers would vary greatly from vendor to vendor. Additional research is required to evaluate data writing to industry-grade programmable logic controllers.

While the experimental results indicate that the overhead of implementing cyber resilience is modest, a realistic SCADA testbed implementation may exhibit increased latency. SCADA environments have rigid time constraints, so the methodology would have to scale to large real-world systems. Also, limited computing resources in industrial SCADA environments would require additional computing equipment to be installed for virtualization. The utilization of Byzantine fault tolerance, which is known for its overhead in terms of communications bandwidth and number of replicas, would exacerbate the implementation challenges in real SCADA environments.

The proposed methodology focuses on a small subset of cyber resilience techniques and improves some characteristics. Moreover, non-persistence introduces new risks. For example, a device that has been repeatedly re-initialized could fail permanently; this risk could be mitigated by having redundant non-persistent devices. Additionally, if virtualization is utilized, the hypervisor should assume the role of monitoring and verifying the successful initialization of non-redundant devices. Layering additional techniques such as distributed allocation and heterogeneity would increase the overall effectiveness. For example, instead of using homogeneous programmable logic controllers, heterogeneous equivalents could be used that would further limit the likelihood of successful cyber attacks. Future research will explore the integration of these and other novel techniques in the cyber resilience methodology.

7. Conclusions

Cyber-resilient systems are becoming increasingly important in the current threat environment. Several resilience techniques are available that can provide useful capabilities in SCADA environments. How-

ever, some techniques, such as non-persistence, are challenging to implement given the tight real-time constraints imposed on SCADA systems. The proposed methodology addresses this challenge by securely storing SCADA system state for use by redundant, non-persistent devices during operation and recovery. The methodology realizes a non-persistent, Byzantine fault-tolerant, virtual industrial control system architecture whose state and function can be stored and restored securely, contributing to its cyber resilience. Implementation of the methodology in a SCADA environment incorporating non-persistent programmable logic controllers reveals that cyber attacks can be identified quickly and secure restoration can occur without loss of state or functionality. The mathematical and timing analyses demonstrate the applicability and efficacy of the methodology in creating cyber-resilient SCADA systems.

Future research will focus on integrating additional resilience techniques such as heterogeneity. Successful application of resilience techniques is vital to mitigating threats to SCADA systems and the critical infrastructure assets they operate.

References

[1] N. Ahmed and B. Bhargava, From Byzantine fault-tolerance to fault-avoidance: An architectural transformation to attack and failure resiliency, to appear in *IEEE Transactions on Cloud Computing*.

[2] K. Astrom and R. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton University Press, Princeton, New Jersey, 2008.

[3] A. Babay, J. Schultz, T. Tantillo and Y. Amir, Toward an intrusion-tolerant power grid: Challenges and opportunities, *Proceedings of the Thirty-Eighth IEEE International Conference on Distributed Computing Systems*, pp. 1321–1326, 2018.

[4] J. Barrowclough and R. Asif, Securing cloud hypervisors: A survey of the threats, vulnerabilities and countermeasures, *Security and Communication Networks*, article no. 1681908, 2018.

[5] F. Bjorck, M. Henkel, J. Stirna and J. Zdravkovic, Cyber resilience – Fundamentals for a definition, in *New Contributions in Information Systems and Technologies, Volume 1*, A. Rocha, A. Correia, S. Costanzo and L. Reis (Eds.), Springer, Cham, Switzerland, pp. 311–316, 2015.

[6] D. Bodeau and R. Graubart, Cyber Resiliency Engineering Framework, MITRE Technical Report MTR 110237, MITRE Corporation, Bedford, Massachusetts, 2011.

[7] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig and S. Sastry, Challenges for securing cyber-physical systems, presented at the *Workshop on Future Directions in Cyber-Physical Systems Security*, 2009.

[8] M. Castro and B. Liskov, Practical Byzantine fault tolerance, *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pp. 173–186, 1999.

[9] Control Solutions Minnesota, Modbus 101 – Introduction to Modbus, St. Paul, Minnesota (`www.csimn.com/CSI_pages/Modbus101.html`), 2020.

[10] B. Cox, D. Evans, A. Filipi, J. Rowanhill, W. Hu, J. Davidson, J. Knight, A. Nguyen-Tuong and J. Hiser, N-variant systems: A secretless framework for security through diversity, *Proceedings of the Fifteenth USENIX Security Symposium*, article no. 9, 2006.

[11] M. Crosby, Nachiappan, P. Pattanayak, S. Verma and V. Kalyanaraman, Blockchain technology: Beyond Bitcoin, *Applied Innovation Review*, vol. 2016(2), pp. 6–10, 2016.

[12] Distributed Systems and Networks Laboratory, Spines, Department of Computer Science, Johns Hopkins University, Baltimore, Maryland (`www.spines.org`), 2020.

[13] G. Engel and M. Mumcouglu, Method for Detecting Anomaly Action within a Computer Network, U.S. Patent No. 0165207 A1, June 12, 2014.

[14] A. Gearhart, P. Hamilton and J. Coffman, An analysis of automated software diversity using unstructured text analytics, *Proceedings of the Forty-Eighth Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, pp. 79–80, 2018.

[15] D. Goodin, FBI tells router users to reboot now to kill malware infecting 500K devices, *Ars Technica*, May 25, 2018.

[16] K. Gordon, M. Davis, Z. Birnbaum and A. Dolgikh, ACE: Advanced CIP evaluator, *Proceedings of the Workshop on Cyber-Physical Systems Security and Privacy*, pp. 90–101, 2018.

[17] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, O'Reilly Media, Sebastopol, California, 2018.

[18] E. Johansson, Virtualization in control systems: Possibilities and challenges, presented at the *SANS European Community SCADA and Process Control Summit*, 2009.

[19] D. Kushner, The real story of Stuxnet, *IEEE Spectrum*, vol. 50(3), pp. 48–53, 2013.

[20] A. Melin, E. Ferragut, J. Laska, D. Fugate and R. Kisner, A mathematical framework for the analysis of cyber-resilient control systems, *Proceedings of the Sixth International Symposium on Resilient Control Systems*, pp. 13–18, 2013.

[21] P. Nachtwey, Feed forwards augment PID control, *Control Engineering*, vol. 52, pp. 42–45, March 31, 2015.

[22] T. Rodrigues Alves, M. Buratto, F. de Souza and T. Rodrigues, OpenPLC: An open-source alternative to automation, *Proceedings of the IEEE Global Humanitarian Technology Conference*, pp. 585–589, 2014.

[23] T. Rodrigues Alves, R. Das and T. Morris, Virtualization of industrial control system testbeds for cyber security, *Proceedings of the Second Annual Industrial Control System Security Workshop*, pp. 10–14, 2016.

[24] R. Ross, M. McEvilley and J. Oren, Systems Security Engineering: Considerations for a Multidisciplinary Approach in the Engineering of Trustworthy Secure Systems, NIST Special Publication 800-160, Volume 1, National Institute of Standards and Technology, Gaithersburg, Maryland, 2016.

[25] J. Sahoo, S. Mohapatra and R. Lath, Virtualization: A survey of concepts, taxonomy and associated security issues, *Proceedings of the Second International Conference on Computer and Network Technology*, pp. 222–226, 2010.

[26] V. Skormin, *Introduction to Automatic Control, Volume I*, Linus Publications, Ronkonkoma, New York, 2009.

[27] J. Szefer, E. Keller, R. Lee and J. Rexford, Eliminating the hypervisor attack surface for a more secure cloud, *Proceedings of the Eighteenth ACM Conference on Computer and Communications Security*, pp. 401–412, 2011.

[28] Technology Transfer Services, The Basics of PLC Operation, *Technology Transfer Blog*, Tampa, Florida (`www.techtransfer.com/blog/basics-plc-operation`), September 9, 2014.

[29] S. Yamamoto, T. Hamaguchi, S. Jing, I. Koshijima and Y. Hashimoto, A hot-backup system for backup and restore of ICS to recover from cyber attacks, in *Advances in Human Factors, Software and Systems Engineering*, B. Amaba (Ed.), Springer, Cham, Switzerland, pp. 45–53, 2016.