ICS-SEA: Formally Modeling the Conflicting Design Constraints in ICS

Eyasu Getahun Chekole
Institute for Infocomm Research, A*STAR
Singapore, Singapore
eyasugc@i2r.a-star.edu.sg

Guo Huaqun
Institute for Infocomm Research, A*STAR
Singapore, Singapore
guohq@i2r.a-star.edu.sg

ABSTRACT

Industrial control systems (ICS) have been widely adopted in mission-critical infrastructures. However, the increasing prevalence of cyberattacks targeting them has been a critical security concern. On the other hand, the high real-time and availability requirements of ICS limits the applicability of certain available security solutions due to the performance overhead they introduce and the system unavailability they cause. Moreover, scientific metrics (mathematical models) are not available to evaluate the efficiency and resilience of security solutions in the ICS context. Hence, in this paper, we propose ICS-SEA to address the ICS design constraints of Security, Efficiency, and Availability (SEA). Our ICS-SEA formally models the real-time constraints and physical-state resiliency quantitatively based on a typical ICS. We then design two real-world ICS testbeds and evaluate the efficiency and resilience of a few selected security solutions using our defined models. The results show that our ICS-SEA is effective to evaluate security solutions against the SEA conflicting design constraints in ICS.

ACM Reference Format:
Eyasu Getahun Chekole and Guo Huaqun. 2019. ICS-SEA: Formally Modeling the Conflicting Design Constraints in ICS. In *Fifth Annual Industrial Control System Security (ICSS) Workshop (ICSS), December 10, 2019, San Juan, PR, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3372318.3372325

1 INTRODUCTION

Industrial control systems (ICS) [24] are well emerged engineering systems where computations and communications are tightly integrated with physical systems to control and monitor a wide range of industrial processes. They are complex systems consisting of multiple interconnected IoT devices such as programmable logic controllers (PLCs), sensors, actuators, human-machine interface (HMI), and various communication devices and protocols.

Nowadays, these systems are being widely adopted by various critical infrastructures, such as smart grids, robotics, transportation, water distribution systems, healthcare services, manufacturing, and so on. Due to their criticalness, these systems are also expected to operate safely, securely, efficiently and in real-time

ICSS, December 10, 2019, San Juan, PR, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7719-5/19/12…$15.00
https://doi.org/10.1145/3372318.3372325

However, their significance in such mission-critical infrastructures makes them to be widely targeted by various types of cyberattacks. The increase in number and diversity of the interconnected smart devices in ICS also widens the attack surfaces. As such, various attack vectors can target ICS at multiple layers. For example, communication-level attacks [9, 12, 16] may exploit vulnerabilities on communication channels and may forge or compromise the integrity of sensor measurements and control commands in transit. On the other hand, memory-safety attacks [6, 10, 22] could also exploit memory-safety vulnerabilities (e.g. buffer overflows and dangling pointers) that could be potentially found in the implementation of certain ICS systems, e.g., the control software or firmware of PLCs. These attacks could corrupt memory of vulnerable programs and hijack or subvert execution flow of certain operations. Therefore, the safety and security of industrial control systems cannot be ignored due to its high significance in many critical systems. Rather, it is a domain that requires urgent and practical security solutions to protect a variety of critical infrastructures at hand.

However, the hard real-time and availability requirements imposed in ICS, alongside the use of resource-constrained computing devices, limits the practical applicability of certain security solutions available. For example, most cryptographic solutions (especially asymmetric key cryptosystems [13, 19]) against communication-level attacks, most memory-safety solutions [4, 17] against memory-safety attacks, access control mechanisms [2, 3] against unauthorized access to critical information, among others, are often considered to be inefficient or impractical in ICS because of their heavy performance overheads. Furthermore, most mitigation strategies against certain cyberattacks often render the system unavailable for a particular or indefinite period of time. For example, mitigation strategies against memory-safety attacks are often based on restarting or aborting the vulnerable system upon attack detection [15, 21]. Mitigation strategies against compromised cryptographic keys often based on revoking keys, isolating nodes or rekeying (which usually causes some service delay). Such mitigation strategies often considered as non-resilient in ICS since the system unavailability for a particular period of time could affect the system dynamics and control-loop stability in ICS. Thus, security, efficiency and availability (SEA) are equally important albeit conflicting design constraints in ICS.

Although the SEA constraints are very critical in the design of ICS, security notions and rigorous treatment of efficiency and availability are not available in the literature. In particular, clear and precise scientific metrics are not proposed to accurately evaluate the efficiency and availability (resilience) of security solutions in ICS (note that availability and resilience might be interchangeably used throughout this paper). In fact, there are several different definitions for *efficiency* and *availability* in the literature, but they are ambiguous and not formally defined to be used in systems with stringent timing constraints, such as ICS. More specifically, there are no mathematical models that accurately quantify what level of overhead or system unavailability (downtime) is tolerable to maintain the system dynamics and control-loop stability of a particular ICS.

Traditionally, efficiency of a security solution is evaluated by benchmarking its overhead with a prior work. Such approach, however, does not often work in the ICS environment where the tolerability of security overheads or system downtime is characterized by various dynamic factors, such as the state information of the physical process at that particular point of time, length of the delay caused by the overhead or system downtime, and the type of control command issued on the prior PLC scan cycle, etc.

In this work, we formally model efficiency and availability (resilience) constraints in ICS based on the real-time and physical-state resiliency requirements of a particular ICS system, respectively. In particular, we follow a control-theoretic approach to quantify the tolerability of delays caused by performance overheads or downtime of services due to certain mitigation strategies. Since the timing constraints can be arisen at various levels in the ICS architecture, we specifically model the efficiency and availability requirements to each individual computing node in ICS such as sensors, PLCs and actuators.

To this end, we design our experimental testbeds based on real-world ICS systems and evaluate the effectiveness of our proposed models. In the future, we expect researchers to leverage our models to evaluate efficiency and resilience of certain security solutions in industrial control systems. We make the following contributions:

- We define and formally model the notions of real-time constraints and physical-state resiliency that are crucial for ICS and should be met alongside strong security guarantees.
- We evaluate the effectiveness of our models on two real-world ICS testbeds involving four practical security solutions.

2 BACKGROUND

In this section, we introduce the relevant background information in industrial control systems. An industrial control system constitutes of complex interactions between entities in the physical-space and cyber-space over communication networks. Unlike mainstream IT systems, such complex interactions are accomplished via communication with the physical-world via sensors and with the digital-world via PLCs (controllers) and other devices. ICS usually impose hard real-time constraints. The underlying system could run into an unsafe and unstable state if such hard real-time requirements are not met. Moreover, the computing devices involved in a typical ICS are also highly resource-constrained. For instance, the sensors, PLCs and other I/O devices in ICS have very limited memory and computational power. Generally, a typical ICS comprises the following components:

- **Physical plant**: The physical system where the actual processes take place.
- **Sensors**: Devices that are capable of reading or observing information from the plant or physical processes.
- **PLCs**: Controller devices that receive sensor inputs and issue control inputs (commands) to actuators.
- **Actuators**: Physical entities that are capable of implementing the control commands they received from the PLCs.
- **Communication networks**: The communication medium through which packets containing sensor measurements, control signals (commands), diagnostic information and alarms transmitted from one ICS component to another.
- **SCADA**: A software designed for monitoring and controlling different processes in an ICS. It often comprises an HMI (human-machine interface) and a historian server. The HMI is used to display the state information of physical plants and processes in the ICS. The historian server is used to store operational data and the history of alarms.

An abstraction of a typical ICS architecture is shown in Figure 1. In Figure 1, x denotes the physical state of the plant, y captures the sensor measurements and u denotes the control command computed by the PLC at any given point of time.

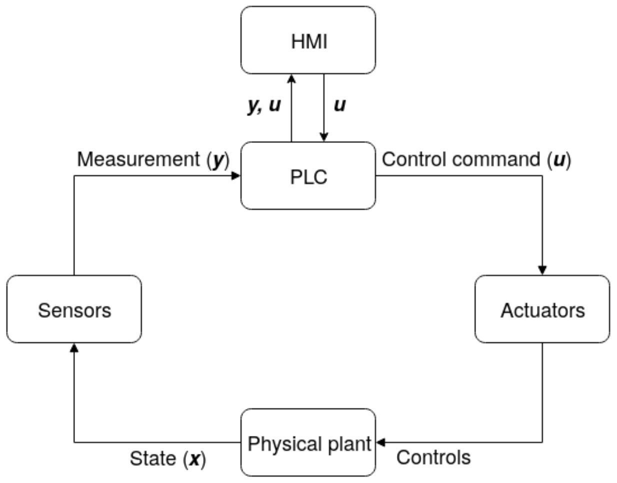

Figure 1: Abstraction of an ICS model

3 MODELING THE ICS DESIGN CONSTRAINTS

Most ICS systems are highly time-critical. The communication between its different components, such as sensors, controllers (PLCs) and actuators, is synchronized by system time. Therefore, delay in these ICS nodes could result in disruption of the control system or damage to the physical plant. In the following sections, we define and discuss the notions of *real-time constraints* and *physical-state resiliency*, which are crucial in the design of ICS and will be used as benchmarks to evaluate the efficiency and resilience of certain security solutions, respectively.

3.1 Modeling real-time constraints

The overall performance of an ICS can be affected by the performance of its individual computing nodes such as PLCs, sensors and actuators. Sensors could take some computation time ($T_y \in \mathbb{R}$) while measuring or observing the state information of physical processes. Similarly, PLCs also take substantial computation time to issue and update control commands ($T_u \in \mathbb{R}$). Actuators could also take some execution time ($T_{act} \in \mathbb{R}$) to implement the control commands received from PLCs (this is often negligible unless the actuator needs to do additional computations, e.g. decrypting the received control command if it was encrypted by the PLC). There will also be communication latencies for packets carrying sensor measurements from sensors to PLCs ($L_y \in \mathbb{R}$) and packets carrying control commands from PLCs to actuators ($L_u \in \mathbb{R}$). Therefore, the overall ICS computation time ($T_{ICS} \in \mathbb{R}$) can be intuitively represented with the following formula:

$$T_{ICS} = T_y + L_y + T_u + L_u + T_{act} \tag{1}$$

Eq. (1) might be used to assess the overall performance of a typical ICS and also to determine a real-time constraint for the overall control loop, accordingly. However, to appropriately synchronize the communication among the ICS nodes as well as to maintain the control dynamics in ICS, real-time constraints are also imposed to each computing node. By design, each ICS node meets their respective real-time constraints. However, when employing certain security solutions, the overhead will be attributed to the computation time of the nodes. Consequently, the introduced overhead might violate the real-time constraint of each node and the ICS system, in general. Although the communication latencies are also critical concerns in ICS, significant latencies are usually a result of communication-level attacks, such as DoS/DDoS attacks. Such issues can be addressed by orthogonal means [25, 26, 28, 29], hence we only focus on the timing constraints at device-level. In the following, we will briefly discuss the real-time constraints imposed in each ICS node.

3.1.1 *Real-time constraints for PLCs.*

PLCs form the main control devices and computing nodes of a typical ICS. As such, PLCs often impose hard real-time constraints to maintain the stability of the control system in ICS.

As shown in Figure 6, PLCs undergo a cyclic process called *scan cycle*. This involves three major operations: input scan, PLC program (logic) execution and output update. The processing time it takes to complete these operations is called *scan time* (T_s). A typical ICS defines and sets an upper-bound on time taken by the PLC scan cycle, called *cycle time* (T_c). This means, the scan cycle must be completed within the duration of the cycle time specified, i.e., $T_s \leq T_c$. We refer this constraint as a *real-time constraint* of the PLC. By design, PLCs meet this constraint. However, due to the additional security overheads introduced to the PLCs (say *PO*), PLCs might not satisfy its real-time constraints. As discussed above, by incorporating a security solution to the PLCs, the scan time increases. This increase in the scan time will be attributed to the PO. Concretely, the introduced overhead can be computed as follows:

$$PO = \hat{T}_s - T_s, \tag{2}$$

where $\hat{T}_s$ and T_s are scan time of the PLC with and without security solution, respectively.

It is crucial to check whether the induced PO still satisfies the real-time constraint imposed to the PLC. To this end, we compute PO for – 1) average-case and 2) worst-case scenarios. Particularly, after incorporating a security solution in the PLC, we compute $\hat{T}_s$ for n different measurements and calculate its respective average-case and worst-case scan time. Formally, we say that the PO is tolerable (i.e. meeting the real-time constraint) in average-case if the following condition holds true:

$$\frac{\sum_{i=1}^{n} \hat{T}_s(i)}{n} \leq T_c \tag{3}$$

Similarly, PO is acceptable in the worst-case scenario if the following condition holds true:

$$\max_{i=1}^{n} \hat{T}_s(i) \leq T_c \tag{4}$$

where $\hat{T}_s(i)$ represents the scan time of the i^{th} measurement after enforcing security.

3.1.2 *Real-time constraints for sensors.*

As discussed in Section 3.1.1, PLCs are required to issue new control commands in every cycle time, i.e., in the interval of T_c. Nevertheless, control commands are issued based on sensor inputs. That means, sensors are required to feed their measurements to their respective PLCs within the interval of the PLC cycle time. Therefore, the real-time constraint of the sensors is also similar to that of their corresponding PLCs, i.e., T_c.

Similarly, sensors respect their real-time constraints by design. However, a security solution implemented to them will introduce an overhead (say *SO*), which might force the sensors not to meet their real-time constraints. Formally, we say that the SO is tolerable if the following condition is satisfied:

$$\hat{T}_y(i) \leq T_c \tag{5}$$

where $\hat{T}_y(i)$ captures the computation time for the i-th sensor measurement after enforcing security.

3.1.3 *Real-time constraints for actuators.*

Actuators are expected to immediately implement the control commands received from PLCs. Since they receive control commands in the interval of the PLCs cycle time, i.e., T_c, they are also expected to implement it in the same interval. If they do not take action within the duration of T_c, the ICS system dynamics will be affected. Therefore, the real-time constraint of the actuators is also the same as their respective PLCs, i.e., T_c. Normally, actuators will meet this constraint by design. However, a security solution implemented to them will add an overhead (say *AO*) which might violate their real-time constraints.

Similarly, AO is tolerable if the following condition holds:

$$\hat{T}_{act}(i) \leq T_c \tag{6}$$

where $\hat{T}_{act}(i)$ captures the computation time of the actuator when implementing the i-th control command, with security.

3.2 Physical-state resiliency

The real-time constraint we discussed in the preceding section plays a crucial role in ensuring a smooth and synchronized communication among the computing nodes in ICS. However, an overhead that does not respect the real-time constraint of a particular node might still be acceptable (with respect to the physical dynamics) if the delay caused by such overhead does not violate the ICS physical dynamics, i.e., causing damage on the physical plant. Moreover, violation of the physical dynamics is often associated with system unavailability (beyond security overhead) for a particular period of time due to various reasons, e.g., due to certain mitigation strategies. In the following, we will discuss the notion of *physical-state resiliency* which is characterized with the physical dynamics of the ICS rather than the user-defined real-time constraints. For the sake of simplifying the presentation, we will focus only on PLCs to define physical-state resiliency, but it also applies to sensors and actuators in a similar way.

The stability of ICS controllers (i.e. PLCs in our case) plays a crucial role in enforcing the dynamics of an industrial control system to be compliant with its requirement. For example, assume that a PLC issues control commands every T_c cycle and an actuator receives these commands at the same rate. Therefore, cycle time of the PLC is T_c. If the PLC is down for an arbitrary amount of time say τ (due to an overhead or the service is terminated for an arbitrary time as part of attack mitigation strategy), then the actuator will not receive fresh control commands for the duration τ. Consequently, the physical dynamics of the respective ICS will be affected for a total of $\frac{\tau}{T_c}$ scan cycles. We note that the duration τ might be arbitrarily large. For example, an existing memory-safety solution [21] (cf. Section 5.1), typically aborts the underlying process or restarts the entire system when a memory-safety attack is detected. Other solutions, e.g., CIMA [8] (cf. Section 5.2), never aborts the underlying system. Nevertheless, CIMA induces an overhead to the scan time of the PLC, as discussed in the preceding section. Consequently, the scan time of PLCs, with CIMA-enabled memory-safety (i.e. $\hat{T}_s$), may increase beyond the cycle time (i.e. T_c). In general, to accurately formulate τ (i.e. the amount of downtime for a PLC), we need to consider the following three mutually exclusive scenarios:

(1) The system is aborted or restarted.
(2) The system is neither aborted nor restarted and $\hat{T}_s \le T_c$. In this case, there will be no observable impact on the physical dynamics of the system. This is because the PLCs, despite having increased scan time, still meet the real-time constraint T_c. Thus, they are not susceptible to downtime.
(3) The system is neither aborted nor restarted and $\hat{T}_s > T_c$. In this scenario, the PLCs will have a downtime of $\hat{T}_s - T_c$, as the scan time violates the real-time constraint T_c.

Based on the intuitions mentioned in the preceding paragraphs, we now formally define τ, i.e., the downtime of a PLC as follows:

$$\tau = \begin{cases} \Delta, & \text{system is aborted/restarted} \\ 0, & \hat{T}_s \le T_c \\ \hat{T}_s - T_c, & \hat{T}_s > T_c \end{cases} \tag{7}$$

where Δ captures a non-deterministic threshold on the downtime of PLCs when the underlying system is aborted or restarted.

Example. As an example, let us consider the first process in SWaT (discussed in Section 4.1). This process controls the inflow of water from an external water supply to a raw water tank. PLC1 controls this process by opening (with "ON" command) and closing (with "OFF" command) a motorized valve, i.e., the actuator, connected with the inlet pipe to the tank. If the valve is "ON" for an arbitrarily long duration, then the raw water tank might overflow, often causing a severe damage to the system. This might occur due to the PLC1 downtime τ, during which, the control command (i.e. "ON") computed by PLC1 may not change. Similarly, if the actuator receives the command "OFF" from PLC1 for an arbitrarily long duration, then the water tank may underflow. Such a phenomenon will still affect the system dynamics. This is because tanks from other processes may expect raw water from this underflow tank. In the context of SWaT, the *tolerability* of PLC1 downtime τ (cf. Eq. (7)) depends on the physical state of the water tank (i.e. water level) and the computed control commands (i.e. ON or OFF) by PLC1. In the next section, we will formally introduce this notion of tolerance, as termed *physical-state resiliency*, for a typical ICS.

To formally model the physical-state resiliency, we will take a control-theoretic approach. For the sake of simplifying the presentation, we will assume that the process dynamics of a typical ICS, without considering the noise and disturbance on the controller, is modeled via a linear-time invariant. This is formally captured as follows (cf. Figure 1):

$$x_{t+1} = Ax_t + Bu_t \tag{8}$$

$$y_t = Cx_t \tag{9}$$

where $t \in \mathbb{N}$ captures the index of discretized time domain, $x_t \in \mathbb{R}^k$ is the state vector of the physical plant at time t, $u_t \in \mathbb{R}^m$ is the control command vector at time t and $y_t \in \mathbb{R}^k$ is the measured output vector from sensors at time t. $A \in \mathbb{R}^{k\times k}$ is the state matrix, $B \in \mathbb{R}^{k\times m}$ is the control matrix and $C \in \mathbb{R}^{k\times k}$ is the output matrix.

We now consider a duration $\tau \in \mathbb{R}$ for the PLC downtime. To check the tolerance of τ, we need to validate the physical state vector x_t at any discretized time index t. To this end, we first assume an upper-bound $\omega \in \mathbb{R}^k$ on the physical state vector x_t at an arbitrary time t. Therefore, for satisfying physical state resiliency, x_t must not exceed the threshold ω. In a similar fashion, we define a lower-bound $\theta \in \mathbb{R}^k$ on the physical state vector x_t.

With the PLC downtime τ, we revisit Eq. (8) and the state estimation is refined as follows:

$$x'_{t+1} = Ax_t + Bu_{t-1}[\![t, t+\tau]\!] \tag{10}$$

where $x'_{t+1} \in \mathbb{R}^k$ is the estimated state vector at time $t+1$ and the PLCs were down for a maximum duration τ. The notation $u_{t-1}[\![t, t+\tau]\!]$ captures that the control command u_{t-1} was active on the actuator for a time interval $[t, t+\tau]$ due to the PLC downtime. In Eq. (10), we assume, without loss of generality, that u_{t-1} is the last control command received from the PLC before its downtime.

Given the foundation introduced in the preceding paragraphs, we say that a typical ICS (cf. Figure 1) satisfies physical-state resiliency if and only if the following condition holds at an arbitrary time index t:

$$\theta \le x'_{t+1} \le \omega$$

$$\theta \le Ax_t + Bu_{t-1}[\![t, t+\tau]\!] \le \omega \tag{11}$$

Figure 2 illustrates three representative scenarios to show the consequence of Eq. (11). If the downtime $\tau_1 = 0$, then u_t (i.e. control command at time t) is correctly computed and $x'_{t+1} = x_{t+1}$. If the downtime $\tau_2 \in (1, 2]$, then the control command u_t will be the same as u_{t-1}. Consequently, x'_{t+1} is unlikely to be equal to x_{t+1}. Finally, when downtime $\tau_3 > 2$, the control command vector u_{t+i} for $i \ge 0$ will be the same as u_{t-1}. As a result, the estimated state vectors x'_{t+j} for $j \ge 1$ will unlikely to be identical to x_{t+j}.

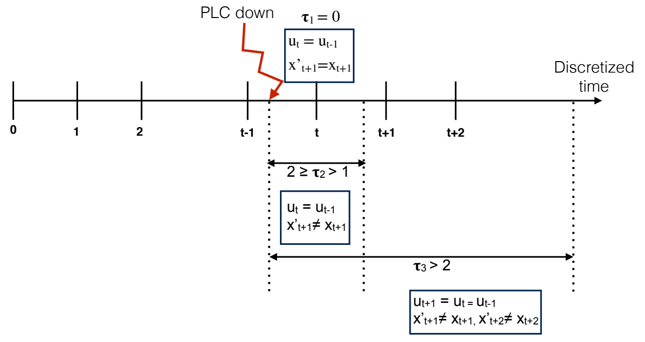

Figure 2: Illustrating the impact of PLC downtime

4 EXPERIMENTAL DESIGN

4.1 SWaT

SWaT [14] is a fully operational water purification testbed for research in the design of secure cyber-physical systems. It produces five gallons/minute of doubly filtered water. In the following, we discuss some salient features and design considerations of SWaT.

4.1.1 Purification process. The entire water purification process is carried out by six distinct, yet co-operative, sub-processes. Each sub-process is controlled by an independent PLC (indexed from PLC1 through PLC6). Specifically, PLC1 controls the first sub-process, i.e., the inflow of water from external supply to a raw water tank, by opening and closing the motorized valve connected with the inlet pipe to the tank. PLC2 controls the chemical dosing process, e.g., water chlorination, where appropriate amount of chlorine and other chemicals are added to the raw water. PLC3 controls the ultrafiltration (UF) process. PLC4 controls the dechlorination process where any free chlorine is removed from the water before it is sent to the next stage. PLC5 controls the reverse osmosis (RO) process where the dechlorinated water is passed through a two-stage RO filtration unit. The filtered water from the RO unit is sent in the permeate tank, where the recycled water is stored, and the rejected water is sent to the UF backwash tank. In the final stage, PLC6 controls the cleaning of the membranes in the UF backwash tank by turning on and off the UF backwash pump. The overall purification process of SWaT is shown in Figure 3.

4.1.2 Components and specifications. The design of SWaT comprises the following components and specifications:

- **Controllers (PLCs)**: six redundancy real-world PLCs (Allen Bradley PLCs) are used to control the entire water purification process. The PLCs communicate one another or with the SCADA system through EtherNet/IP or common industrial protocol (CIP).
- **Remote input/output (RIO)**: SWaT also consists of remote input/output terminals containing digital inputs (DI), digital outputs (DO) and analog inputs (AI) for each PLC. The RIO of SWaT consists of 32 DI (water level and pressure switches), 13 AI (water pressure, flow rate and water level sensors), and 16 DO (actuators such as motorized valves and pumps).
- **PLC program**: SWaT has a complex PLC program (control software) written in ladder logic. It comprises various instructions such as boolean operators, arithmetic operators, comparison operators, conditional statements, counters, timers, contacts, and coils (the full list is provided in Table 1). The most complex PLC of SWaT (i.e. PLC2) has a PLC program containing 129 instructions.
- **SCADA system**: an HMI is mounted to SWaT to provide users with a local system supervisory, control and monitoring. It also displays state information of plants, sensors, actuators and operational status of PLCs to users.
- **Operation management**: consisting of historian server (to store operational data, various events and alarm histories) and engineering workstation (designed to provide all necessary control graphics).
- **Real-time constraint**: the real-time constraint, i.e., cycle time, of SWaT is 10ms. The notion of cycle time and real-time constraint (in the ICS context) is briefly presented in Section 3.
- **Communication frequency**: the PLCs in SWaT communicate one another and also with the HMI depending on certain operational conditions. Considering the most complex PLC (with 129 instructions), it sends as many as 382 packets per second to its most active peer or as few as three packets per second to another peer. Considering connections with all devices in SWaT, we estimate that the most complex PLC has a send over receive request ratio of 1000 packets per second.

Concurrently, SWaT is based on closed-source and proprietary Allen Bradely PLCs. Hence, it is not possible to directly modify the firmware of these PLCs and to enforce memory-safety solutions. To alleviate this problem in our experimental evaluation, an open platform, named Open-SWaT, was designed.

4.2 Open-SWaT

Open-SWaT is a mini ICS we designed using OpenPLC controller [18] – an open source PLC designed for industrial control and cyber-physical systems. Figure 4 shows a high-level architecture of the design of Open-SWaT. With Open-SWaT, we reproduce features and operational behaviours of SWaT. In particular, we reproduce the main factors that have substantial effect on the scan time of PLCs. Some salient features of the Open-SWaT design are discussed as follows.

(1) **PLCs**: The PLCs of Open-SWaT are designed using OpenPLC controller that is hosted on a Raspberry PI device and runs on Linux operating system. To replicate the hardware specifications of the PLCs in SWaT, we configured 200MHz fixed CPU frequency and 2Mb user memory for the PLCs used in Open-SWaT.
(2) **Remote Input/Output (RIO)**: Arduino Mega has been used as the RIO terminal in Open-SWaT. It has an AVR-based processor with a clock rate of 16MHz. It comprises 86 I/O pins where different I/O devices can be directly connected to them. To replicate the number of input and outputs in SWaT, we used 32 digital inputs (DI) (switches, push-buttons and scripts), 13 analog inputs (AI) (ultrasonic and temperature

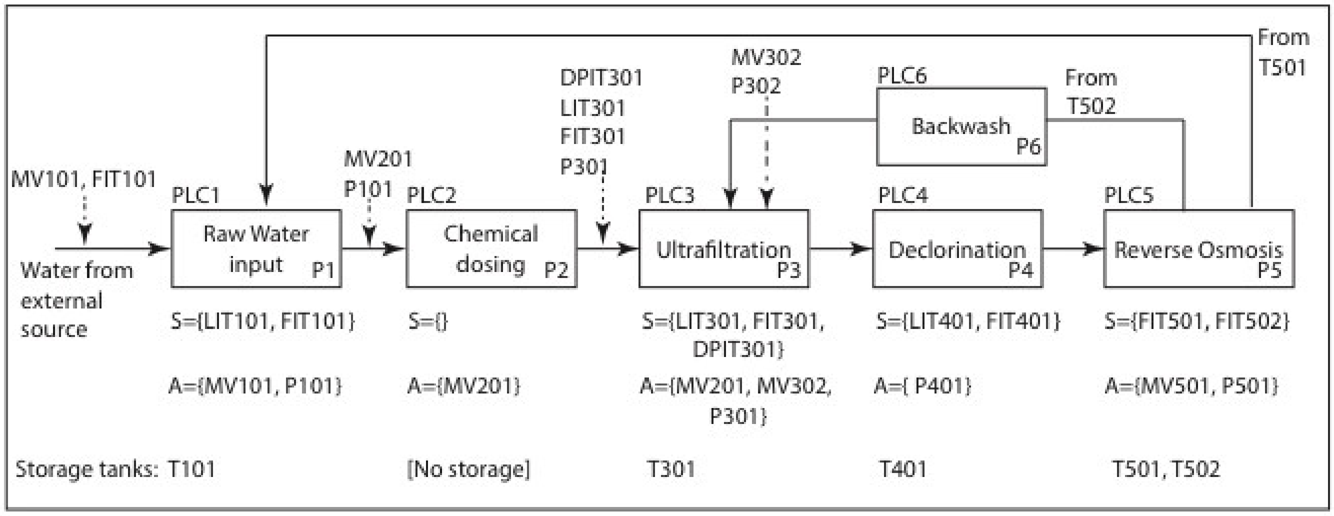

Figure 3: Overview of the water purification process of SWaT [1]
Definition of the acronyms: S = Sensor, A = Actuator, T = Tank, P = Process, MV = Motorized Valve, LIT = Level Indicator Transmitter, FIT = Flow Indicator Transmitter, DPIT = Differential Pressure Indicator Transmitter.

sensors) and 16 digital outputs (DO) (light emitter diodes or LEDs, in short).

(3) **PLC program**: We wrote a control software (control logic) using ladder logic. It is designed to have similar complexity with the control logic in SWaT. A sample of the logic diagram is depicted in Figure 5. The control program consists of several types of instructions such as 1) *logical:* AND, OR, NOT, SR (set-reset latch); 2) *arithmetic*: addition (ADD), multiplication (MUL); 3) *comparisons:* equal-to (EQ), greater-than (GT), less-than (LT), less-than-or-equal (LE); 4) *counters:* up-counter (CTU), turn-on timer (TON), turn-off timer (TOF); 5) *contacts:* normally-open (NO) and normally-closed (NC); and 6) *coils:* normally-open (NO) and normally-closed (NC). The overall control program consists of 129 instructions in total; details are shown on Table 1.

(4) **Communication frequency**: The communication architecture of Open-SWaT (as illustrated in Figure 4) consists of analogous communicating components with that of SWaT. Open-SWaT uses both types of modbus protocols – modbus TCP (for wired or wireless communication) and modbus RTU (for serial communication). The communication among PLCs is via modbus TCP or modbus RTU whereas the communication between PLCs and the SCADA system is via modbus TCP. Frequency of communication among PLCs and the SCADA system is similar to that in SWaT. The communication between PLCs and Arduino is via the USB serial communication. The communication frequency between Arduino and sensors is 100Hz.

(5) **Real-time constraint**: Since the cycle-time (real-time constraint) of SWaT is 10ms, we also set 10ms cycle time to each PLC in Open-SWaT.

(6) **SCADA system**: We use ScadaBR [20], a full SCADA system consisting of web-based HMI, for Open-SWaT.

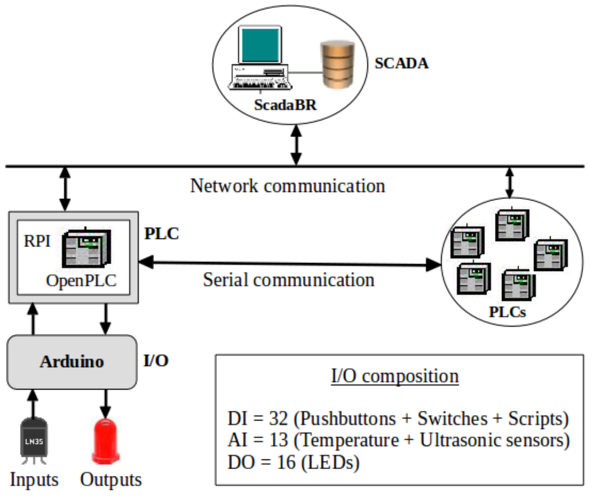

Figure 4: Architecture of Open-SWaT

4.3 SecUTS

The Secure Urban Transportation System (SecUTS) is a ICS testbed designed to research on the security of a Metro SCADA system. The Metro SCADA system [30] comprises an *Integrated Supervisory Control System (ISCS)* and a *train signaling system.* ISCS integrates localized and centralized control and supervision of mechanical and electrical subsystems located at remote tunnels, depots, power substations and passenger stations. The entire Metro system can be remotely communicated, monitored, and controlled from the operation control center via the communication network. On the other hand, the signaling system facilitates communications between train-borne and track-side controllers. It also controls track-side equipments and train position localization. Modbus is used as a

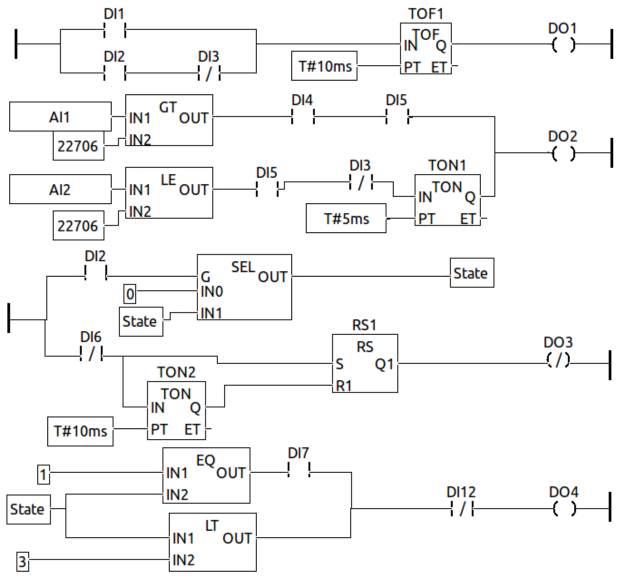

Figure 5: Sample PLC program in ladder diagram

Table 1: Complexity of the SWaT PLC program

Instruction(s)	Type	count
AND	Logical	17
OR	Logical	14
NOT	Logical	5
SR	Logical	1
ADD	Arithmetic	1
MUL	Arithmetic	2
EQ , GT	Comparison	6
LT , LE	Comparison	4
TON	Timers	3
TOF	Timers	9
CTU	Counters	1
SEL, MAX	Selection	2
NO	Contacts	38
NC	Contacts	3
NO	Coils	21
NC	Coils	2
Total		129

communication protocol among the devices in the ISCS. A detailed account of the Metro SCADA can be found in [30].

The SecUTS testbed provides facilities to examine several types of cyber attacks, such as message replay, forged message and memory-safety attacks, in the ISCS system and enforce proper countermeasures against such attacks. However, the SecUTS testbed is also based on closed-source proprietary Siemens PLCs, hence we cannot directly enforce security solutions to these PLCs. Consequently, we similarly designed **Open-SecUTS** testbed (by mimicking SecUTS) using OpenPLC controller. It consists of 6 DI (emergency and control buttons) and 9 DO (tunnel and station lightings, ventilation and alarms). The control system in SecUTS is less delay sensitive and its cycle time is 30ms.

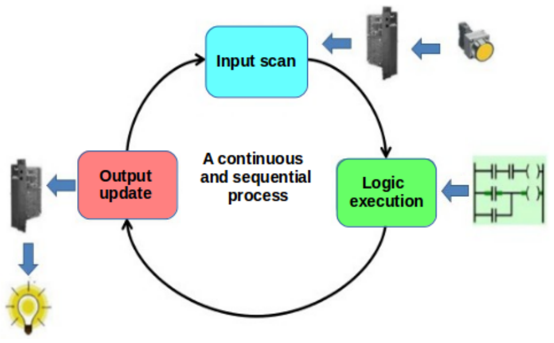

Figure 6: The scan cycle of a PLC

5 THE SECURITY SOLUTIONS UNDER TEST

To assess the practicality and effectiveness of our proposed models, we evaluate the efficiency and resilience of certain security solutions using the proposed models. To this end, we choose four security solutions, comprising two memory-safety and two cryptography solutions. Our choice is motivated by the heavy performance overhead often introduce by memory-safety and cryptographic solutions. In addition, we were also interested to involve different levels of security solutions in the test. As such, we choose system-level (i.e. memory-safety) and communication-level (i.e. cryptography) security solutions. In the following, we provide a high-level description of the security solutions we choose for the test.

5.1 ASan

AddressSanitizer (ASan) [21] is a compile-time code instrumentation memory-safety tool. It inserts memory-safety checks into the program code at compile-time, and it detects memory-safety violations at runtime. ASan instruments C/C++ programs at compile time. The instrumented program will then contain additional ASan libraries, which are checked to detect possible memory violation at runtime. Such an instrumented code can detect buffer overflows/underflows, use-after-free errors (dangling pointers, use-after-return errors, initialization order bugs and memory leaks. As a mitigation strategy, ASan simply aborts the system whenever a memory-safety violation or an attack is detected, which renders the system unavailable for an indefinite period of time.

Since ASan was primarily designed for x86 architectures, it has compatibility issues with ARM-based architectures. Thus, for the purpose of this experiment, we adapted ASan for ARM-based architecture in our Open-SWaT testbed. We empirically measured its run-time overheads and evaluate its efficiency and resilience in a realistic ICS environment (cf. Section 6).

5.2 CIMA

CIMA (Countering Illegal Memory Accesses) [8] is a resilient mitigation strategy we recently developed against memory-safety attacks. It is developed on top of ASan aiming to address the mitigation limitation of ASan. CIMA mitigates memory-safety attacks by proactively bypassing illegal memory access instructions, i.e., instructions that attempt to access memory illegally. It is based on stopping an attack before the attack actually exploit the vulnerability. CIMA never aborts the vulnerable program when mitigating memory-safety violations, hence favoring availability of the system.

5.3 2FA

2FA (two-factor authentication) [11] is a secure communication protocol based on two-factor authentication developed for a Metro Control system. In this protocol, historical data from a server is used as a second factor, in addition to a secret key, to authenticate a server communicating with the controller (PLC).

This protocol was tested in the SecUTS (cf. Section 4.3) testbed and its performance overhead was measured. We evaluated its efficiency and resilience using our proposed models (cf. Section 6).

5.4 LCDA

LCDA (legacy-compliant data authentication) [5] is a cryptographic authentication method developed for an industrial control system. The main goal is to verify authenticity of communication between PLCs in ICS and to assess the efficiency of such solutions in resource-constrained legacy systems. In this work, the authors benchmarked symmetric and asymmetric signature algorithms for a variety of hardware platforms. This solution was experimented on the SWaT testbed (cf. Section 4.1).

6 ANALYSIS AND EVALUATION

In this section, we discuss the evaluation of the selected security solutions using our proposed models on the Open-SWaT and Open-SecUTS testbeds. In particular, we evaluate the *efficiency* – tolerability of the runtime overheads introduced to each security solution, and the *resilience* – capability of the security solutions to ensure system availability and maintain physical-state resiliency in ICS even in the presence of cyberattacks. However, evaluating the security guarantee of these security solutions, e.g., detection and mitigation accuracy, is out of the scope of this work.

6.1 Efficiency

6.1.1 ***ASan and CIMA.*** With our ICS environment integrated in the Open-SWaT and Open-SecUTS, the average memory-safety overhead (MSO) induced by ASan is 53.46% and 50.4%, respectively. Additionally, CIMA induces 8.06% and 6.53% runtime overheads on Open-SWaT and Open-SecUTS, respectively. Thus, the overall runtime overhead of the combined security measure is 61.52% (for Open-SWaT) and 56.93% (for Open-SecUTS). A more detailed performance report for ASan and CIMA, including the performance overhead of each PLC operation in both testbeds, is illustrated on Table 2 and 3.

It is crucial to check whether the induced overhead by ASan and CIMA ($\hat{T}_s$) is acceptable in the ICS environment. To this end, we evaluate if this overhead respects the *real-time constraints* of SWaT and SecUTS. For instance, consider the tolerability in average-case scenario. We observe that this security measure satisfies the condition of tolerability, as defined in Eq. (3). In particular, from Table 2, $mean(\hat{T}_s)$ = 441.72µs, and T_c = 10ms; and from Table 3, $mean(\hat{T}_s)$ = 398.39µs, and T_c = 30ms. Consequently, Eq. (3) is satisfied and the overhead induced by ASan and CIMA is both tolerable in SWaT and SecUTS testbeds.

Similarly, considering the worst-case scenario, we evaluate if Eq. (4) is satisfied. From Table 2, $max(\hat{T}_s)$ = 3167.15µs, and T_c = 10ms; and from Table 3, $max(\hat{T}_s)$ = 2506.39µs, and T_c = 30ms. It is still tolerable, thus the proposed security measure satisfies SWaT's and SecUTS's real-time constraints in both scenarios. Therefore, despite high security guarantees provided by ASan and CIMA, its overhead is still tolerable in a ICS environment.

6.1.2 ***2FA.*** The 2FA protocol introduces an overhead on the PLC ranging from 18ms to 25ms when tested on different number of historical data. This overhead is tolerable since the PLC cycle time (i.e. T_c) of the SecUTS testbed is 30ms.

6.1.3 ***LCDA.*** The performance overhead of LCDA was measured using different hardware platforms and cryptographic algorithms. The overhead varies when using different combinations of hardware platforms and cryptographic protocols over various packet sizes. As such, the overhead was tolerable at one time and it was not at another time.

6.2 Resilience

6.2.1 ***ASan.*** As discussed, ASan simply aborts the vulnerable program when a memory-safety attack is detected, hence it renders system unavailability. That means, $\tau = \infty$, hence ASan does not satisfy the physical-state resiliency requirement in ICS.

6.2.2 ***CIMA.*** In Section 6.1.1, we showed that the overall overhead is tolerable, i.e., $\hat{T}_s \leq T_c$. Hence, the additional overhead induced by CIMA does not affect the physical-state resiliency. Added to the fact is that the availability of SWaT and SecUTS is also ensured by CIMA since it never aborts the system or leads to PLC downtime when mitigating memory-safety attacks. That means, the downtime, i.e., τ, is zero, hence Eq. (11) is satisfied. This then ensures physical-state resiliency.

6.2.3 ***2FA.*** As discussed in Section 6.1.2, the overhead of 2FA is tolerable with respect to the real-time constraint of SecUTS. As such, the introduced overhead does not affect the physical-state resiliency of SecUTS. The 2FA protocol does not also render system unavailability since it is just an authentication protocol and does not involve any mitigation strategy. As such, the system downtime (τ) is zero. Therefore, the 2FA security solution meets the physical-state resiliency requirement of SecUTS.

6.2.4 ***LCDA.*** LCDA is also an authentication mechanism and does not render the system unavailable. While the performance overhead in some platforms is in acceptable range, it is very high other platforms. As such, the real-time constraint as well as the physical-state resiliency of SWaT is not met in some combinations of hardware platforms and cryptographic protocols.

7 RELATED WORK

Since the conflict between security, efficiency and resilience has been critical in the context of ICS, the need to define new security notions and a rigorous treatment of efficiency and availability is high. However, to the best of our knowledge, there is no prior work that clearly defines and models these critical design conflicts in ICS.

Table 2: Memory-safety overheads for the Open-SWaT Testbed

Operations	Number of cycles	Network devices	CPU speed (in MHz)	Original (T_s)		ASan				ASan + CIMA ($\hat{T_s}$)			
				Mean (in µs)	Max (in µs)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)
Input scan	50000	6	200	59.38	788.12	118.44	1132.32	59.09	99.46	122.86	1151.35	63.48	106.9
Program exec.	50000	6	200	69.09	611.82	115.88	720.36	46.79	67.72	118.97	802.18	49.88	72.2
Output update	50000	6	200	145.01	981.09	185.37	1125.45	40.36	27.83	199.89	1213.62	54.88	37.85
Full scan time	50000	6	200	273.48	2381.03	419.69	2978.13	146.21	53.46	441.72	3167.15	168.24	61.52

Table 3: Memory-safety overheads for the Open-SecUTS Testbed

Operations	Number of cycles	Network devices	CPU speed (in MHz)	Original (T_s)		ASan				ASan + CIMA ($\hat{T_s}$)			
				Mean (in µs)	Max (in µs)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)	Mean (in µs)	Max (in µs)	MSO (in µs)	MSO (in %)
Input scan	50000	1	200	59.84	739.94	114.88	902.01	55.04	91.98	115.07	906.09	55.23	92.3
Program exec.	50000	1	200	48.56	488.38	91.36	443.61	42.8	88.14	104.41	676.19	55.85	115.01
Output update	50000	1	200	145.47	850.62	175.59	1045.34	30.12	20.71	178.91	924.11	33.44	22.99
Full scan time	50000	1	200	253.87	2078.94	381.83	2390.96	127.96	50.4	398.39	2506.39	144.52	56.93

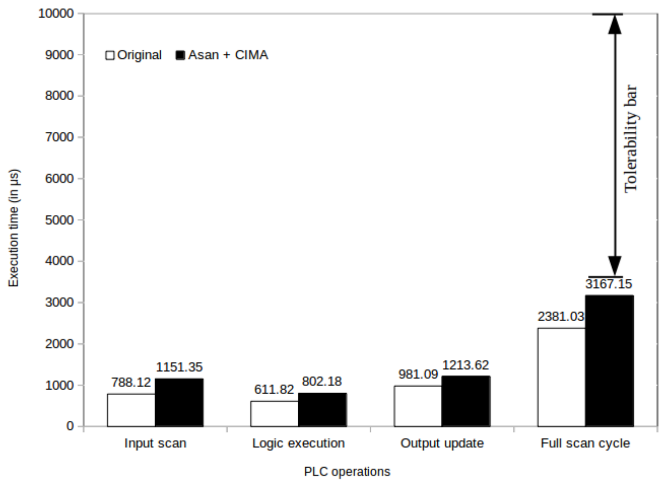

Figure 7: Tolerability of the worst-case overhead for Open-SWaT testbed

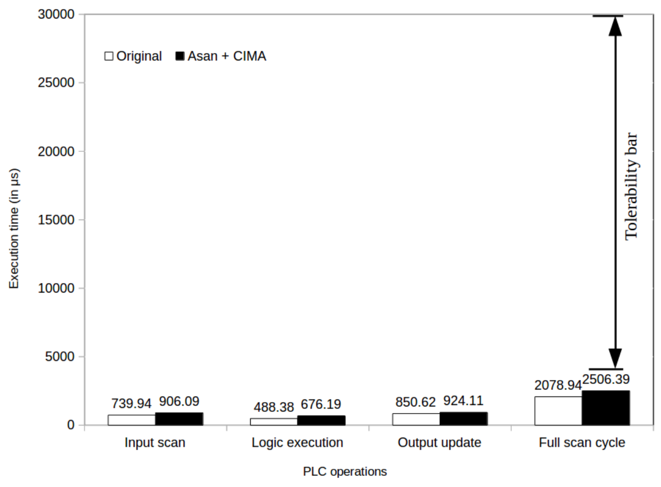

Figure 8: Tolerability of the worst-case overhead for Open-SecUTS testbed

Zhang et al. [27] tried to model the trade-off between privacy and performance in a cyber-physical system (CPS) context. While they leveraged the differential privacy approach to preserve privacy of CPS, they also analyzed and modeled its performance overhead. They proposed an approach that optimizes the system performance while preserving privacy of CPS. This work is interesting from point of view of analyzing performance overheads in CPS, but did not provide any analysis with respect to the efficiency and resilience constraints in CPS or ICS.

Stefanov et al. [23] proposed a new model and platform for the SCADA system of an integrated CPS. With the proposed platform, they modeled real-time supervision of CPS, performance of CPS based on communication latencies, and also he assessed communication and cyber security of the SCADA system. He followed a generic approach to assess and control various aspects of the CPS. However, he did not specifically work on the efficiency and resilience requirements in ICS or CPS.

Chekole et al. [6, 7] tried to empirically evaluate the tolerability of memory-safety overheads in a CPS environment, but did not model the real-time and physical-state resiliency constraints using the dynamic factors involved in the ICS system dynamics.

Castellanos et al. [5] empirically evaluated acceptability of an overhead caused by a cryptographic solution in an ICS environment. However, his analysis only focuses on the number of packets transmitted when using different hardware platforms and cryptographic algorithms.

8 CONCLUSION

In industrial control systems, security remains a critical concern. However, the hard real-time and availability requirements, which are often conflicting with security solutions, are also equally critical. As such, balancing these conflicting design constraints will remain a critical challenge in the design of industrial control systems.

In this paper, we formally model the efficiency (i.e. the real-time) and availability (i.e. the resilience) constraints imposed in a typical industrial control system. Using these models, we evaluate the efficiency and resilience of some security solutions on two realistic ICS testbeds. We also expect other researchers to use our models as benchmarks when evaluating efficiency and resilience of their security solutions in ICS.

From a conceptual point of view, this work provides a fresh outlook over the critical design constraints in industrial control systems and it might inspire the community to propose more accurate models in this regard. In future, we plan to extend this work by considering further factors in the ICS system dynamics. We also intend to enhance the proposed efficiency and resilience models by systematically correlating and consolidating the specific requirements at each individual node level.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation (NRF), Prime Minister's Office, Singapore, under its National Cybersecurity R&D Programme (Award No. NRF2014NCR-NCR001-31) and administered by the National Cybersecurity R&D Directorate.

REFERENCES

[1] Chuadhry Mujeeb Ahmed, Jianying Zhou, and Aditya P. Mathur. 2018. Noise Matters: Using Sensor and Process Noise Fingerprint to Detect Stealthy Cyber Attacks and Authenticate Sensors in CPS. In *Proceedings of the 34th Annual Computer Security Applications Conference (ACSAC'18)*. 566–581.

[2] Alessandro Armando, Roberto Carbone, Eyasu Getahun Chekole, Claudio Petrazzuolo, Andrea Ranalli, and Silvio Ranise. 2014. Selective Release of Smart Metering Data in Multi-domain Smart Grids. In *Smart Grid Security*, Jorge Cuellar (Ed.). Springer International Publishing, Cham, 48–62.

[3] Alessandro Armando, Roberto Carbone, Eyasu Getahun Chekole, and Silvio Ranise. 2014. Attribute Based Access Control for APIs in Spring Security. In *Proceedings of the 19th ACM Symposium on Access Control Models and Technologies (SACMAT '14)*. ACM, New York, NY, USA, 85–88. https://doi.org/10.1145/2613087.2613109

[4] Nathan Burow, Derrick McKee, Scott A. Carr, and Mathias Payer. 2018. CUP: Comprehensive User-Space Protection for C/C++. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS'18)*. ACM, New York, NY, USA, 381–392. https://doi.org/10.1145/3196494.3196540

[5] John Henry Castellanos, Daniele Antonioli, Nils Ole Tippenhauer, and Martín Ochoa. 2017. Legacy-Compliant Data Authentication for Industrial Control System Traffic. In *Applied Cryptography and Network Security*.

[6] E. G. Chekole, J. H. Castellanos, M. Ochoa, and D. K. Y. Yau. 2017. Enforcing Memory Safety in Cyber-Physical Systems. In *Katsikas S. et al. (eds) Computer Security. SECPRE 2017, CyberICPS 2017*. https://doi.org/10.1007/978-3-319-72817-9_9

[7] E. G. Chekole, S. Chattopadhyay, M. Ochoa, and G. Huaqun. 2018. Enforcing Full-Stack Memory Safety in Cyber-Physical Systems. In *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS'18)*.

[8] Eyasu Getahun Chekole, Unnikrishnan Cheramangalath, Sudipta Chattopadhyay, Martín Ochoa, and Huaqun Guo. 2018. Taming the War in Memory: A Resilient Mitigation Strategy Against Memory Safety Attacks in CPS. *CoRR* abs/1809.07477 (2018). arXiv:1809.07477 http://arxiv.org/abs/1809.07477

[9] M. Conti, N. Dragoni, and V. Lesyk. 2016. A Survey of Man In The Middle Attacks. *IEEE Communications Surveys Tutorials* 18, 3 (thirdquarter 2016), 2027–2051.

[10] A. Francillon and C. Castelluccia. 2008. Code Injection Attacks on Harvard-architecture Devices. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*.

[11] Huaqun Guo, Ezekiel Wei Ren Tan, Luying Zhou, Zhigang Zhao, and Xingjie Yu. 2019. 2FA Communication Protocol to Secure Metro Control Devices. In *The IEEE Intelligent Transportation Systems Conference (ITSC)*. Auckland, New Zealand.

[12] R. M. Góes, E. Kang, R. Kwong, and S. Lafortune. 2017. Stealthy deception attacks for cyber-physical systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. 4224–4230.

[13] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL Landscape: A Thorough Analysis of the x.509 PKI Using Active and Passive Measurements. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference (IMC '11)*. ACM, New York, NY, USA, 427–444. https://doi.org/10.1145/2068816.2068856

[14] iTrust. 2017. Secure Water Treatment (SWaT) testbed. https://itrust.sutd.edu.sg/research/testbeds/secure-water-treatment-swat/.

[15] KASAN. 2018. The Kernel Address Sanitizer. https://www.kernel.org/doc/html/v4.12/dev-tools/kasan.html.

[16] Amit Kleinmann, Ori Amichay, Avishai Wool, David Tenenbaum, Ofer Bar, and Leonid Lev. 2018. Stealthy Deception Attacks Against SCADA Systems. In *Computer Security*, Sokratis K. Katsikas, Frédéric Cuppens, Nora Cuppens, Costas Lambrinoudakis, Christos Kalloniatis, John Mylopoulos, Annie Antón, and Stefanos Gritzalis (Eds.). Springer International Publishing, Cham, 93–109.

[17] G. C. Necula, J. Condit, M. Harren, S. McPeak, and W. Weimer. 2005. CCured: Type-safe Retrofitting of Legacy Software. *ACM Trans. Program. Lang. Syst.* (2005).

[18] OpenPLC. 2018. OpenPLC. http://www.openplcproject.com/.

[19] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126. https://doi.org/10.1145/359340.359342

[20] ScadaBR. 2018. ScadaBR. http://www.scadabr.com.br/.

[21] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov. 2012. AddressSanitizer: a fast address sanity checker. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'12)*.

[22] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.R. Sadeghi. 2013. Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization. In *S&P'13*.

[23] A. Stefanov, C.-C. Liu, M. Govindarasu, and S.-S. Wu. 2015. SCADA modeling for performance and vulnerability assessment of integrated cyber-physical systems. *International Transactions on Electrical Energy Systems* 25, 3 (2015), 498–519.

[24] Daniel Sullivan, Eric Luiijf, and Edward J. M. Colbert. 2016. Components of Industrial Control Systems. *Advances in Information Security Cyber-security of SCADA and Other Industrial Control Systems* (2016), 15–28. https://doi.org/10.1007/978-3-319-32125-7_2

[25] J. C. M. Teo, L. H. Ngoh, and H. Guo. 2009. An Anonymous DoS-Resistant Password-Based Authentication, Key Exchange and Pseudonym Delivery Protocol for Vehicular Networks. In *2009 International Conference on Advanced Information Networking and Applications*. 675–682. https://doi.org/10.1109/AINA.2009.65

[26] E. N. Ylmaz, B. Ciylan, S. Gönen, E. Sindiren, and G. Karacayılmaz. 2018. Cyber security in industrial control systems: Analysis of DoS attacks against PLCs and the insider effect. In *2018 6th International Istanbul Smart Grids and Cities Congress and Fair (ICSG)*.

[27] H. Zhang, Y. Shu, P. Cheng, and J. Chen. 2016. Privacy and performance trade-off in cyber-physical systems. *IEEE Network* 30, 2 (2016), 62–66.

[28] L. Zhou and H. Guo. 2017. Applying NFV/SDN in mitigating DDoS attacks. In *TENCON 2017 - 2017 IEEE Region 10 Conference*. 2061–2066. https://doi.org/10.1109/TENCON.2017.8228200

[29] Luying Zhou, Huaqun Guo, and Gelei Deng. 2019. A fog computing based approach to DDoS mitigation in IIoT systems. *Computers & Security* 85 (August 2019), 51–62. https://doi.org/10.1016/j.cose.2019.04.017

[30] Luying Zhou, Huaqun Guo, Dong Li, Jun Wen Wong, and Jianying Zhou. 2017. Mind the Gap: Security Analysis of Metro Platform Screen Door System. In *Proceedings of the Singapore Cyber-Security RandD Conference (SG-CRC'17)*.