UNIVERSITY OF CALIFORNIA
SANTA CRUZ

VIRTUALIZED ENVIRONMENTS TO ANALYZE CYBER-PHYSICAL ATTACKS AND DEFENSES

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Luis Salazar

September 2024

The Dissertation of Luis Salazar
is approved:

Alvaro Cardenas, Chair

Chen Qian

Cormac Flanagan

Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

Abstract

Virtualized environments to analyze cyber-physical attacks and defenses

by

Luis Salazar

The ever-increasing ubiquity of cyber-physical systems is creating an attractive target for developing malware and cyber attacks. From consumer-grade IoT devices like drones to specialized industrial equipment in critical infrastructure, many new and varied attacks and malware emerge to afflict these systems, impacting the physical environment and processes handled by the devices. A fundamental part of protecting a system against malware and cyber attacks involves understanding the nature of the attack to devise an effective countermeasure. However, due to the nature of some of these systems, a comprehensive analysis might be unfeasible or outright dangerous. To overcome this limitation, we propose using a virtualized scenario, emulating the system's behavior, more specifically the physical and networking behavior, to dynamically analyze cyber-attacks and malware without compromising the physical integrity of the systems and their environment. We first test the feasibility of this idea by simulating cyber-attacks against a virtual quadcopter drone before testing the attacks on its physical counterpart once the attacks are deemed "safe" for the user and the drone. Then, we emulate an industrial process using software-defined networks to evaluate the feasibility of implementing defense mechanisms against attacks with some features provided by software-defined networks. Finally, we combine the virtualization of cyber-physical systems and the use of software-defined networks to simulate a power grid system to dynamically analyze the behavior of an actual malware known as "Win32.Industroyer" by infecting a virtual machine in an isolated virtual environment. Our ultimate goal is to refine a framework to allow researchers to safely

simulate cyber-physical systems to test attacks, defenses, and malware against a virtual avatar of the existing system.

To my grandfather, who saw a century of advancements

and inspired me to look forward.

May he rest in peace.

Augusto Salazar Sánchez (1921 - 2023)

Acknowledgments

First and foremost, I want to thank my advisor, Dr. Alvaro Cardenas, who continuously guided and supported my efforts. I count myself as lucky to have had him as my advisor, for it is truly rare to find someone with his dedication and discipline. I learned a lot from him, and I will be forever grateful.

I would also like to thank the members of my reading committee for their valuable time, effort, patience, and consideration—Dr. Chen Qian, Dr. Cormac Flanagan, and Dr. Ricardo Sanfelice, who joined my advancement exam.

I want to thank my labmates, both from UT Dallas and UCSC, for their valuable camaraderie and insights. Kelvin Mai and Raul Quinonez went above and beyond to welcome me. Xi Qin showed me the ropes and opened my eyes to new insights. Neil Ortiz is a good friend and a dependable colleague with whom I enjoyed several conversations, on and off-topic. Juan Lozano always has an interesting viewpoint. Sebastian Castro is a like-minded person, and I always enjoy our discussions. Luis Burbano has a very meticulous demeanor and always offers his unbiased opinion. Diego Ortiz and Juanita Gomez always seem to see something I missed and offer valuable feedback.

I want to thank my parents and siblings, who always believed in me and supported me in every endeavor. Who instilled my values and principles in me and always encouraged me to take the next step.

Last but not least, Theo-Alyce Gordon, the department's academic advisor, has always been incredibly helpful.

Chapter 1

Introduction

Cyber-physical systems, interconnected by a data network and interacting with the physical world, are becoming more widespread, making them an appealing target for cyberattacks and malware. These attacks can affect various devices, from consumer-grade IoT devices like drones to specialized industrial equipment. As a result, an increasing number of attacks and malware varieties are being developed to target these systems, causing damage to the physical environment and processes that the devices handle.

It is essential to understand the potential threats affecting different systems to ensure their protection. However, there are situations where it is not feasible or safe to conduct attack scenarios due to the dangerous consequences of a successful attack, such as hazardous conditions or significant financial loss. For instance, a successful cyberattack against industrial processes can result in life-threatening situations, depending on the physical process. Another example is that, against drone systems, a successful compromise of the underlying system can pose a risk of injury to bystanders.

This plight has become increasingly dangerous among systems that provide essential services to society, part of our civilization's critical infrastructure. These

attacks are threatening enough when carried out by motivated individuals, such as the Maroochy Shire attack by a disgruntled former employee [9]. Even more so when governments sponsor nation-state actors to potentially coordinate cyber attacks with military operations, such as the multiple Russian operations against Ukraine targeting the power grid and heating systems [34, 44, 8, 11, 12, 21].

These threats beg the question of how you should study these attacks and malware without affecting the systems you intend to protect. Some of these systems are critical, and a recovery process from a catastrophic failure, such as a power grid black-start [40], can delay the operation for days. In other cases, tests against cyber-physical systems can expose bystanders to injury, for instance, if a successful attack against a drone forces it to plummet to the ground. Therefore, we thought of using virtualized systems to safely test such dangerous or unfeasible scenarios.

This research aims to develop a comprehensive framework to emulate a cyber-physical system's physical and networking elements. It will provide a virtualized environment that enables researchers to conduct detailed analyses of existing threats and test various attack and defense mechanisms. The framework will allow for precise simulations of complex scenarios, providing valuable insights into the vulnerabilities and strengths of cyber-physical systems. By facilitating a safer and more efficient testing environment, this research will play a critical role in enhancing the security and resilience of these systems against cyber attacks.

To ensure the feasibility of our new solution, we devised a plan to assess the overall idea. First, we intend to use existing simulations to develop attacks on a system from a virtualized environment. This is an effective way to test how the system will respond to potential attacks and the fidelity of the available simulation. Once we develop the attack, we will test it in the physical device,

within a controlled environment, to check for any vulnerabilities. Our focus will be on Autonomous Vehicles (AVs), as several different simulators are available for various AVs.

After determining whether the idea is feasible, we assess a crucial component essential to our solution: Software-Defined Networks (SDN). We intend to utilize the adaptable nature of SDN as part of the virtualization framework to simulate the networking behavior of the cyber-physical systems within the framework. We want to determine if SDN features are suitable enough to simulate a cyber-physical system and test a reasonable countermeasure that leverages this component. Having done that, we now have the two primary components of a cyber-physical system in a virtual surrogate: the physical simulation and the networking component. However, these components rely on existing simulators. Ideally, a comprehensive framework must be extensible and allow custom simulations.

The next step is to identify the requirements for the framework and develop a solution that meets those requirements. To define the criteria, we consider a threat against a sophisticated cyber-physical system where testing is not feasible. To illustrate this, we will focus on the power grid, a prime target for cyber-attacks and strategically important in international conflicts, thus considered a critical infrastructure. Moreover, real-world examples of nation-state cyber attacks against a power grid exist. Our intuition is that if the solution can adapt to this intricate scenario, it suffices as a reasonable solution to conduct security research on other less elaborate cyber-physical systems.

Once we have a solid framework in place, we continue to enhance it with the goal of outsmarting both malware and human attackers. We add extra features to create a hybrid-interaction honeypot, which simulates interaction with a non-existent physical system, making the attacker believe they are engaging with a

real process.

1.1 Outline of the Dissertation

In this section, we present the outline of this dissertation and a summary of each chapter. We begin the dissertation by introducing some fundamental concepts useful to understanding and establishing a context for the remainder of the document in chapter 2. We then present related efforts considering the main approaches of our study in chapter 3. In chapter 4, we clarify the particular research problem we are interested in, formulate the scope, and explain the methodology and research questions. In chapters 5 to 9, we endeavor to answer our research questions by delivering the results from our attack simulations, software-defined network evaluation, framework design, malware study, and framework enhancements.

Chapter 1: Introduction - Cyber-Physical Systems security challenges. In this chapter, we present the primary motivation of the study, showcasing the unique challenges these systems have and how using virtualized surrogates can aid researchers in this endeavor. We define various approaches to illustrate the advantages of virtualized systems.

Chapter 2: Background - We provide fundamental information relevant to the context of the various approaches we take to develop the virtualization framework. We delve into details of autonomous vehicles, software-defined networks, and the power grid.

Chapter 3: Related work. We first indicate the primary research trends we will focus on for the remainder of the study, showcasing relevant efforts driving these topics. We start by presenting various works related to threats against autonomous vehicles, followed by various security solutions for cyber-physical sys-

tems, and ending with different power grid simulations and models.

Chapter 4: Proposed Work - Defining our research scope, methodologies, and threat model for the research questions. In this chapter, we want to construct a systematic way to address the various virtualizations in accordance with specific assumptions regarding the attacker and define minimal requirements for our simulated environments.

Chapter 5: We showcase the experiments and results regarding our efforts to use virtualized versions of a drone to test different attacks, comparing the results between the virtualized version and the actual drone, showcasing the feasibility of using a virtualized surrogate as a viable replacement of a cyber-physical system.

Chapter 6: We showcase the experiments and results regarding our efforts to showcase how the various features of software-defined networks can be fundamental tools in security research. To accomplish this, we test a defense mechanism and evaluate its performance in a virtualized environment.

Chapter 7: We showcase the design of our simulation framework. From the design challenges to the requirements, we present our considerations when designing the virtualization framework. We aim to use this tool to study a real-world malware sample in a contained, virtualized environment.

Chapter 8: We showcase the results of our study regarding the infamous malware Industroyer. We analyze both iterations of the malware using our simulation framework, observing the behavior of each malicious payload to gain previously unknown insights about the artifacts.

Chapter 9: We demonstrate viable ways to enhance the framework for diverging lines of research, showcasing its flexibility and extensibility. We add honeypot functionalities to our framework, increasing the veracity of the simulation.

Chapter 2

Background

This chapter presents some introductory information regarding the primary topics and key concepts we will address to further the reader's understanding: Autonomous vehicles, software-defined networks, and a brief description of the power grid.

2.1 Autonomous vehicles

Autonomous Vehicles (AVs), including aerial, ground, and sea vehicles, are becoming integral to our lives. Unmanned aerial vehicles are projected to be the most popular kind of AV with a projected 11.2 billion dollar global market by 2020 [55] and with applications ranging from agricultural management to aerial mapping and freight transportation [28].

AVs have a variety of sensors at their disposal to interact with the physical environment, ranging from cameras to GPS and Inertial Measurement Units (IMU). An IMU is a standard component in AVs and includes *accelerometers*, *gyroscopes*, and *magnetometers*. Accelerometers measure a vehicle's acceleration, gyroscopes measure a vehicle's angular velocity, and magnetometers act like a compass for

the vehicle. A typical configuration includes one accelerometer, gyroscope, and magnetometer per vehicle axis. The three axes are *pitch* (rotating a vehicle upwards or downwards), *roll* (rotating the vehicle sideways), and *yaw* (rotating the orientation of the vehicle). In addition to IMUs, AVs typically use sensors like GPS receivers for location information, RADARs, LiDARs, or ultrasonic sensors to detect nearby obstacles and cameras.

The vehicle's autopilot processes all this sensor information to complete a mission. In a simulated environment, the simulator takes the position of the sensors and actuators, reflecting any changes within a simulated scenario. To exchange sensor information and rotor output, the autopilot and the simulator send and receive messages encapsulated within the MAVLink network protocol [50]. The simulator sends messages with the sensor data (GPS, accelerometer, gyroscope, magnetometer, etc.); the autopilot sends the actuator control messages.

The MAVLink protocol defines messages encapsulated in variable-length packets sent through a TCP connection or UDP data frames (figure 2.1). The packet starter field determines the protocol version. The flags fields are only available in version 2.0. The message ID is 1 byte for versions 0.9 and 1.0 and 3 bytes for version 2.0. The signature is only available in version 2.0 and is optional. The particular autopilot we used employs a TCP connection to communicate with the ground controller and a UDP data stream to handle internal communications.

2.2 Software-Defined Networks

In the traditional networking switch architecture, the various functions that a device must fulfill are segregated into three main categories represented as layers or planes: management, control, and data. Peer communications occur in the same plane, whereas cross-category communications occur between planes. Figure 2.2

1 byte	:: **Packet starter**
1 byte	:: **Payload length**
1 byte	:: **Incompatibility flags**
1 byte	:: **Compatibility flags**
1 byte	:: **Sequence number**
1 byte	:: **System ID**
1 byte	:: **Component ID**
1|3 bytes	:: **Message ID**
0-255 bytes	:: **Payload**
2 bytes	:: **Checksum (X.25 CRC)**
0-13 bytes	:: **Signature (optional)**

Figure 2.1: MAVLink protocol frame format.

illustrates the most common interactions between these categories.

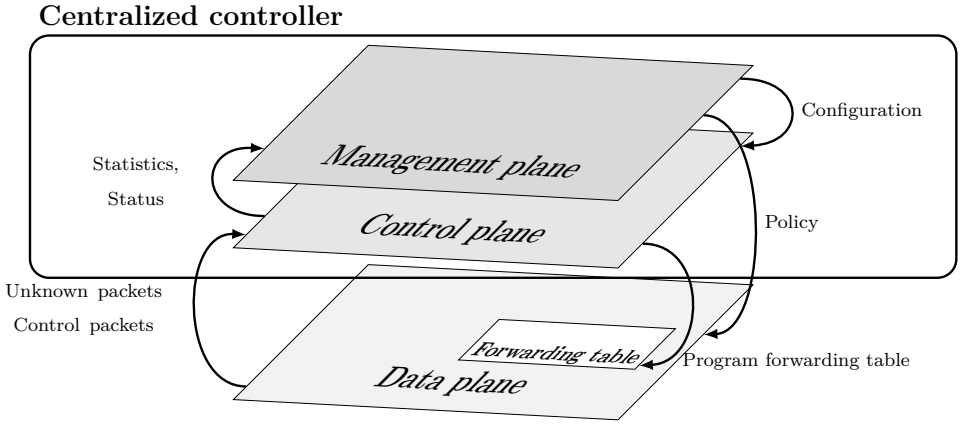


Figure 2.2: SDN centralized controller planes.

The main idea behind SDN is to move all the software involving the decision-making process out of the network switches and into a centralized system to optimize decisions based on a complete overview of the managed network and not just a single device. Ultimately, SDN leaves the forwarding responsibilities to the

device and centralizes the decision-making responsibilities within a controller.

In most scenarios, most packets handled by an SDN switch involve the data plane, which manages packet buffering, scheduling, and forwarding. Suppose the forwarding table has not recorded the header information of any given packet, or the received packet is a control packet. In that case, these packets are delivered to the control plane to obtain the corresponding table entry. The remaining features involve the interaction between the switch and the administrator, handled by the management plane. In short, the data plane handles the raw data to be forwarded by the device, and the intelligence regarding the decision-making process is carried out by the control plane based on the configuration made by the administrator through the management plane.

2.2.1 Southbound API

Since a remote controller takes care of the decision-making process, there has to be a mechanism with which the network switches communicate with said controller. This communications channel is the southbound API, which handles the messages between the controller and the SDN devices to configure the network. The Open Networking Foundation developed OpenFlow as its standard southbound API and is supported by several manufacturers such as Cisco, Juniper, Huawei, Brocade, IBM, Dell, and HP.

In addition to developing OpenFlow, the Open Networking Foundation published the OpenFlow switch specification [15], describing the actual requirements that a logical SDN switch must have to properly support OpenFlow as a southbound API. These requirements include the support of OpenFlow as the communications protocol between the switch and controller and include the main components of an OpenFlow-compliant Switch. The main components of an OpenFlow

logical switch can be summarized in a set of flow tables, a group table, and a set of OpenFlow channels with the controller.

By using OpenFlow as a southbound API, the controller can add, delete, or update flow entries within the tables in a logical device, both in a predefined manner or dynamically, in response to particular conditions in the packets.

A flow table is a set of flow entries at a high level. Each entry comprises match fields and a set of instructions or actions to carry out for each matching packet. This matching is carried out in priority order, starting from the first table within the switch and checking each subsequent table for a matching entry, taking into account the actual priority of every entry. If there is a matching entry, the switch follows the instructions associated with that entry; if there is no one, the switch follows the instructions configured in the "table-miss" entry.

A critical characteristic of a flow entry involves the *actions* the switch will carry out for every matching packet (e.g., output port = x). However, the OpenFlow specification states that this action set can only contain one action of each type. This restriction means that if the administrator wants to execute the same action type several times with different parameters for the exact matching packet, one entry is insufficient, which poses a problem since only the first matching entry will be executed, excluding any further actions contained in additional matching flow entries with lower priority.

To address this restriction, the OpenFlow specification defines the group table comprising group entries. Figure 2.3 depicts the main idea behind group tables. Every group entry contains an ordered list of action buckets, where each action bucket is essentially an action set of a regular flow entry. This feature enables an administrator to execute multiple action sets for a matching packet. By stating a specific group as the action within the action set of a flow entry, the matching

packet will be processed with the actions described in the action buckets of the corresponding group entry.

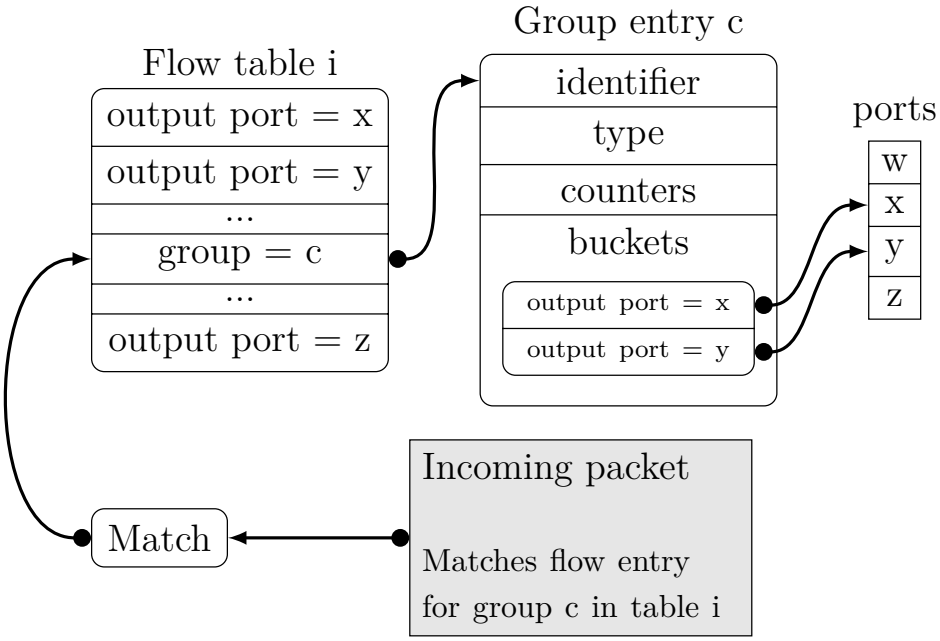


Figure 2.3: An example of a group table action.

Figure 2.3 shows an example of a group table action as part of a flow table entry. An incoming packet that matches the entry on the flow table i that has an action set including the group c will instruct the switch to find the c entry within its group table and execute the actions defined in the buckets of the group entry, which allows the execution of multiple actions of the same type for a single matching packet.

2.2.2 Northbound API

In addition to having switches communicating with the controller, we need a way to let network applications reach the controller. The main idea behind the northbound API is to provide a mechanism with which an application can communicate with the controller to manage the network. The rationale behind this

idea is that an application might handle a different set of information regarding the process implemented on top of a network that might not be directly related to network packets (e.g., sensor data, business analytics, market behavior, natural phenomena). With such information, different network decisions might be more effective toward the objective of the process, whatever it might be.

Due to the potential diversity of such applications and the fact that they might not be implemented within the controller, the latter must provide different mechanisms to fulfill such requirements. That is where the northbound API comes into play.

Different controllers implement different types of northbound APIs. However, most controllers show a trend in how they implement these APIs. For the most part, controllers implement at least two different APIs that achieve the same goals. The main API provides extensions in the same programming language used to build the controller (i.e., Python API, Java API).

However, controller developers usually implement a secondary API, which provides an external communications channel between an application and the controller via some specific protocol. The northbound API is traditionally implemented by providing a REST API for this communications channel.

2.2.3 The SDN controller

The SDN controller is the centralized system that makes all the forwarding decisions in an SDN. It is this system that becomes crucial in any SDN implementation. There are several controllers, some of which are suitable for quick prototypes, while others are fully developed commercial-grade controllers.

Currently, the two most popular open-source SDN controllers are OpenDaylight and ONOS. The Open-Source Foundation supports the former, while the

Open Networking Foundation supports the latter. While both controllers are Java-based and provide several desirable features, the same organization that defines the OpenFlow standard specifications maintains ONOS, which guarantees seamless compatibility with the protocol. This difference becomes evident in the performance variations between the two controllers. While both controllers are comparable, ONOS has been proven to outperform OpenDaylight[63]. For these reasons, we have decided to implement our solution in ONOS.

2.3 The power grid

The primary purpose of a power grid is to supply the energy that users demand. However, because storing energy in high amounts is not feasible, any generated power must be consumed while it is being generated. Therefore, power grids demand coordination between supervision and control, where *Supervisory Control and Data Acquisition (SCADA)* systems fulfill an essential role, making them the primary target for an attack.

A complete power grid comprises generation, transmission, and distribution. In generation are the power plants and the substations that increase the voltage to the transmission level. In transmission, there are mainly the power lines and intermediate substations that help keep the voltage level during the long journey to the consumer. Finally, the distribution systems are the substations that decrease the voltage to consumption level and distribute the electricity between end-users.

The SCADA system supervises and controls the power grid's operation within acceptable levels for the elements in the power grid. However, power grid operation is still complex and requires human decision-making in contingency cases. SCADA are systems whose inputs on one side are measurements (e.g., voltage, amperage, and breaker positions) and output the operation state of the system. On the

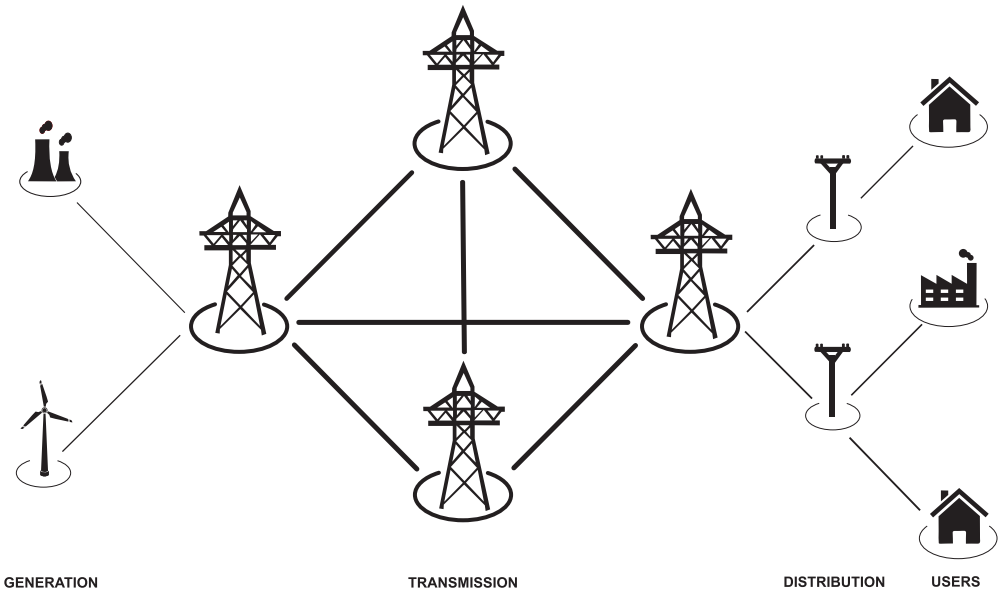


Figure 2.4: Power Grid Components

other hand, inputs are commands that produce topology change actions over the system.

Many critical infrastructures, such as power systems, have existed for over a century; however, it is only in the past two decades that remote monitoring and control of these systems migrated from serial communications to IP-compatible networks, supporting various industrial control protocols such as IEC 60870-5-104 (IEC 104), DNP 3.0, Modbus/TCP, ICCP, and IEC 61850. Some protocols focus on substation automation (IEC 61850), others on communications between control centers (ICCP), and others on remote monitoring and control of large-scale systems (IEC 104 or DNP 3.0).

An RTU is an electronic device that interfaces the power grid with the SCADA. These devices collect the measurements in the substations and report them to the SCADA system using one of the remote monitoring and control communication protocols. Aside from that, they execute the commands received from the SCADA. All the communication between SCADA and RTU is carried out in compliance

with several standards. Among them is the standard IEC 60870-5-104.

2.3.1 IEC 60870-5-104 (IEC 104)

One of the most recent cyberattacks against power grid systems occurred in 2016 in Ukraine, where attackers issued control commands via the IEC 104 protocol to cause a power outage that affected more than 200,000 consumers [44].

The International Electrotechnical Commission (IEC) originally developed IEC 104 in 1995, and its original version is IEC 60870-5-101 (IEC 101). Then, in 2000, it was extended to IEC 104. It is widely used in Europe and Asia to monitor and control large geographical areas. IEC 104 encapsulates telecontrol messages into Application Protocol Data Units (APDUs) over TCP/IP using port 2404, as shown in figure 2.5.

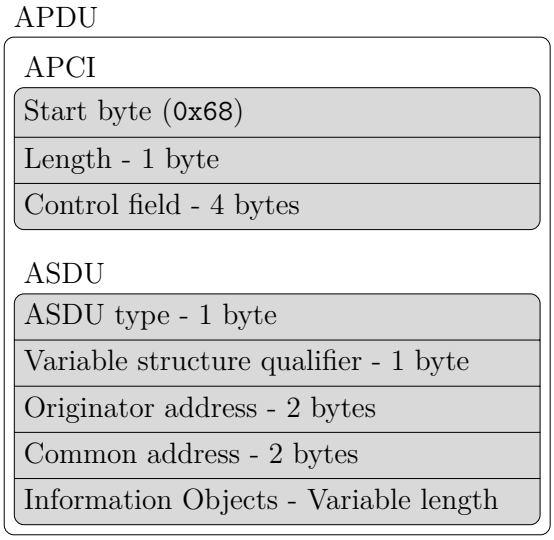


Figure 2.5: IEC 60870-5-104 frame.

The contents of this TCP payload have one or more APDUs. The *Application Protocol Control Information (APCI)* is the first part of an APDU, which acts as the message's header. The *Application Service Data Unit (ASDU)* is the second

part. It carries the sensor values and control messages between the controlled station (e.g., *Remote Terminal Unit (RTU))* and the controller station (e.g., SCADA systems).

There are three types of APDUs:

I-Format APDUs carry sensor and control data between endpoints.

S-Format APDUs acknowledge after a specific (but configurable) number of I-format APDUs received.

U-Format APDUs provide three connection control functions: start transmission (`STARTDT act/con`), stop transmission (`STOPDT act/con`) and keepalive connection request (`TESTFR act/con`).

An ASDU is an I-Format APDU that is comprised of a data unit identifier (DUI) and an information object (IO), as shown in figure 2.5. DUIs always maintain the same structure for all types of ASDUs, while the structures of IOs vary for different ASDUs. DUIs involve "what" kind of data or command is being sent (type ID) and "why" it was sent (cause of transmission). Each IO corresponds with a specific field device, and the device has a unique ID known as the Information Object Address (IOA). IOs encapsulate the actual measurement data values or control commands within their IOAs.

2.3.2 Manufacturing Message Specification (MMS)

Manufacturing Message Specification (MMS) is a Transmission Control Protocol (TCP) in the IEC 61850 stack that supports real-time process data and supervisory control information between network devices and computer applications. MMS supports communication between client and server devices to monitor

and control the different objects. The client can be a system or a monitoring device, and the server represents the objects accessed by the client. These objects are shielded in a Virtual Manufacturing Device (VMD), which includes domains and variables to execute operations such as read, write, event signaling, etc. The MMS protocol delivers information from the substation Bay Level to the Station Level, exchanging data of the Logical Nodes and their components. That information varies depending on each logical node, primarily used to send data to higher-level devices such as RTUs and SCADA systems, predominantly used inside a substation.

Communication in MMS (broadly IEC 61850) is modeled as object-oriented classes and methods. Every physical device in the network can be accessed using its network address. Logical Devices (LDs), such as bay control, protection relay, breaker IED, etc., are associated with the physical device. It comprises several Logical Nodes (LNs) that define different functionalities of the actual devices. Examples of Logical nodes are Measurement (MMXU), which provides voltage and current per phase, Control Switch (CSWI), Breaker Switch (XSWI), Circuit Breaker (XCBR), etc. The properties of every logical node are defined using the functional constraints (FCs), data objects (DOs), and data attributes (DAs). This data is mapped onto the TCP/IP protocol, allowing the client to access the server through an IP address to read/write data and exchange files.

Chapter 3

Related work

Our work is related to several lines of research. For our first approach and feasibility study of the simulation, our work intersects with existing attacks against autonomous vehicles. There are also various defense mechanisms involving cyber-physical systems, potentially allowing us to mimic or incorporate them in a software-defined network. Finally, our ultimate objective of building a convincing simulation to analyze sophisticated malware involves existing simulations and models.

3.1 Threats against autonomous vehicles

Most autonomous vehicles trust all sensor data to make navigation and other control decisions. In addition, they trust that the control command given to actuators is executed faithfully. While the trusted sensor and actuator data without any form of validation have proven to be an effective trade-off in current market solutions, it is not a sustainable practice as AVs become more pervasive and sensor attacks mature in their sophistication.

There are two main threats to autonomous vehicle sensors: *GPS spoofing* and

transduction attacks. GPS spoofing attacks have happened in real-world systems. For example, several instances of GPS spoofing attacks affecting the navigation of more than 24 vessels in the Black Sea have been reported [70] (and experts believe these GPS attacks are anti-drone measures). Moreover, while there is skepticism to believe the claims that Iran spoofed a military-grade GPS to capture a U.S. Unmanned Aerial Vehicle [52], launching the same takeover attack on commercial GPS systems is relatively easy [29, 26, 65].

Another set of attacks against autonomous vehicles (and any cyber-physical device) is *transduction attacks*. Sensors are transducers that translate a physical signal into an electrical one. Still, these sensors sometimes have a coupling between the property they want to measure and another analog signal the attacker can manipulate. For instance, sound waves can affect accelerometers and compel them to report incorrect movement values [66], and radio waves can trick pacemakers into disabling pacing shocks [31]. These attacks are effective on drones by using sound to affect gyroscopes [61] or lasers to affect camera sensors [10].

Classical security mechanisms, such as software security, memory protection, authentication, or cryptography, are not enough to protect these cyber-physical systems, as transduction attacks represent a new class of attacks that classical software security does not effectively handle [18]. To identify these latest attacks, there is a growing interest in Physics-Based Attack Detection (PBAD) [20].

PBAD consists of two steps. The first step extracts the system's physical invariants to create a model to predict future sensor measurements. One of the most popular models of physical invariants is called a Linear Dynamic State-Space Model (LDS) [67]. The second step for PBAD is an online anomaly detection algorithm that compares predictions with observed states and warns the operator when the accumulated discrepancy between predicted and observed states exceeds

a threshold. PBAD has been studied in water control systems [24], state estimation in the power grid [37], boilers in power plants [69], chemical processes [5, 7], and a variety of other cyber-physical systems [20].

3.2 Security solutions for cyber-physical systems

As far as we could identify, the most prominent defense mechanism for cyber-physical systems in the current literature is honeypots. To gather information about novel threats to ICS, honeypots can be used to identify new attack patterns to help us prepare better countermeasures. The design of general-purpose ICS honeypots has many challenges, including the diversity of vendors, industrial protocols, physical processes, control devices, and functionalities. Previous work has faced challenges when trying to replicate the diversity of these systems while ensuring a high-fidelity environment.

The application of honeypots in CPS is a growing area of research. Irvene et al. present an example of a honeypot for a robotic vehicle [27], where they fool an attacker into believing that a given attack was successful by simulating unsafe actions within a honeypot environment of the protected robotic system. For ICS, Rubio et al. [53] present, among other intrusion detection solutions, a commercial use case of a honeypot solution developed for acquiring and analyzing information related to a threat or attack against an ICS. Another relevant example developed under the Honeynet project is Conpot[1]: a low interactive server-side ICS honeypot that provides a range of typical industrial control protocols. While Conpot presents examples of industrial systems to attackers, a sophisticated attacker can identify the honeypot by observing that the system does not satisfy the traditional physical properties of devices in a typical industrial setting. To further

[1] `http://conpot.org`

enhance the realism of the simulated system, Litchfield et al. present HoneyPhy, a honeypot that models the physics of the devices (e.g., delays) [36].

In addition to honeypots, our work is related to SDN applied to CPS. Skowyra et al. [58] showed how a verifiably safe SDN can be implemented as part of a CPS. With the versatility of such networks, new ideas emerge involving this concept of programmable networks as part of a CPS and, more precisely, as part of an ICS. Antonioli et al. developed MiniCPS, a research tool that leverages the flexibility of SDN to present a framework with which any researcher can simulate an entire ICS network corresponding to a physical model of a known system [4].

3.3 Power grid simulations and models

Malware analysis is more effective if it can be deployed in a sandbox that represents a realistic environment where it is intended to be deployed. A securely isolated virtual machine is sufficient to understand general malware affecting system resources. However, we must emulate or simulate new equipment and industrial protocols to interact with the malware. A cost-effective solution is to use a simulated environment replicating the conditions of such critical infrastructures, allowing us to observe both the malware's digital behavior and the attack's physical repercussions.

Some simulation software frameworks, such as Mosaik [56], simulate various scenarios in large-scale smart grids. According to the smart grid network topology, its core function is to create a communication framework among the network endpoints. With this communication framework, all the components, including the control centers and Remote Terminal Unit (RTU), can exchange the control commands and the measurement data in the corresponding physical control system events. Mosaik configures TCP socket connections for the simulated power

grid components to connect to Mosaik modules. However, it does not emulate the configurations in the network layers of the OSI model. Thus, users have no settings for specifying industrial protocols and other network configurations. As a co-simulation framework, Mosaik requires other established simulators (e.g., Matlab Simulink) to model the smart grid components. It provides APIs to connect to the external power grid simulator to access these models.

We evaluated some existing power grid testbeds to determine whether they would be a suitable starting point for our purposes (Table 3.1). We examined whether the system provides an appropriate device and network components simulation, is fully open-source, easily extensible, and supports all the protocols typically used in power grid systems.

Table 3.1: Power grid / Smart Grid testbeds.

	[41]	[6]	[39]	[42]	[54]	[25]	[57]	[1]	[23]
Physical simulation	●	●	●	○	○	●	○	○	●
SITL / HITL	●	○	●	●	●	●	●	●	●
Network simulation	●	●	●	●	●	●	○	●	●
Proprietary modules	●	●	●	●	●	●	●	●	●
Industrial protocols	0/4	0/4	0/4	2/4	2/4	2/4	1/4	1/4	1/4

○: Not supported ●: Supported

While searching for different testbeds and simulators, we noticed that the ones with an accurate power grid model lacked the capabilities to support the required network protocols (in particular, none supported IEC-101 and IEC-104), and those with accurate network simulations lacked a reliable physical model. Most of them have some external component, such as a system-in-the-loop (SITL) or hardware-in-the-loop (HITL), making it difficult to replicate or use proprietary modules that prevent it from being easily extensible, as its original goal constraints the design. HITL testbeds aim to test and refine a system by adding hardware components

that reflect the changes in the physical system. In contrast, SITL testbeds add an existing system to simulated real-time devices to test and refine these simulations.

These environments rely on proprietary software or hardware, limiting their extensibility and openness. In addition, none of them support all the devices or the industrial protocols we need in our study.

Chapter 4

Proposed work

4.1 Problem statement

Exploring cybersecurity within Cyber-Physical Systems (CPS) presents an exciting and crucial challenge. Integrating computational elements with physical processes in CPS creates unique difficulties, making safe and controlled research impractical. Given the vital role of CPS in critical infrastructure like power grids, any experimentation carries significant risks. To develop secure and resilient systems, we need innovative research frameworks that accurately simulate these complex systems without compromising their integrity or safety.

4.2 Scope

In this dissertation, we propose establishing a methodology to develop accurate virtual representations of cyber-physical systems to test or analyze malware and cyberattacks against such systems safely.

The roadmap to develop such a methodology starts by evaluating the feasibility of properly executing attacks against a commercial drone device's existing virtu-

alized development framework. Once the attacks have been tested and deemed reasonably safe, we proceed to test the same attacks on the physical counterpart, the commercial drone. For an attack to be deemed "safe", the scenario in which it is executed must not endanger any bystanders or the drone itself. Moreover, the attacks must only affect the drone without inadvertently compromising any nearby cyber-physical device.

After testing the feasibility of virtualizing attacks, we evaluate the possibility of using dynamic virtualized elements in a network to determine the effectiveness of new defenses against known attacks directed at cyber-physical infrastructures. Once again, we use existing virtualized technologies to implement the new defense mechanisms to test these ideas.

In conclusion to the feasibility of implementing attacks and defenses in a virtualized environment, we propose developing a modular emulation framework capable of reproducing the physical characteristics of a power grid to analyze the behavior of malware, specifically Win32/Industroyer and its newer variant Industroyer2. As this malware interacts with the target system using specialized protocols, and no viable virtualized system to date can effectively interact with such malware, we develop our emulation framework compliant with the necessary industry standards.

We identified the following criteria for our solution:

Isolation: It must be isolated from any other network.

Secure-coupling: We must be able to integrate a machine infected with malware.

Network tracking: A network traffic capture is essential to understand how the malware interacts with its targets.

Flexibility: It must support various industrial control protocols. In this specific instance, power grid protocols for remote substation control and substation automation.

Physical simulation: It must simulate changes in a physical process.

Customizable: The simulation scenarios must be configurable.

Extensible: It must support future scenarios with different processes.

Once the framework is effective against a real threat, we will test its extensibility by adding additional features to provide new functionalities that are not necessarily required to interact with the malware but aid in convincing an attacker that they are interacting with a real system. Honeypot functionalities.

4.3 Threat model

In all of the evaluated scenarios, there is going to be a consistent threat actor for each case. In every scenario, the attacker has already partially compromised the network, allowing them to communicate with the target device. That is, the attacker already has a foothold within the network with the necessary privileges to communicate with the target. In the case of the autonomous vehicle, this means access to the wireless network used to communicate with its autopilot. In the remaining cases, a workstation within the monitoring network is assumed to be compromised.

Moreover, we assume the attacker has detailed knowledge of the target system. In the case of autonomous vehicles, they know the details of the vehicle's sensors, actuators, models, and autopilot. In the other cases, they know the target IP

addresses, device identifications, where applicable, and target object identification, where applicable.

While this threat may seem somewhat overestimated at first glance, this is where the capabilities the attackers had during the cyberattacks against Ukraine's power grid. Thus, we are accommodating our threat model to observe the same attacker.

4.4 Research questions

Following the previously mentioned scope, we intend to answer the following research questions:

- Is it possible to seamlessly replicate an attack for a cyber-physical system in a virtualized environment?
- Do the simulated consequences represent a reasonable abstraction of the real world?
- Can a newly devised cyberattack be tested before safely replicating it against a physical device?
- Can virtualized systems aid in the defense of a targeted system?
- Is it possible to implement a virtual defense mechanism that would remain undetected?
- How feasible is it to develop a realistic virtualized environment to replicate complex industrial control systems?
- Is it possible to safely interact with real malware under strictly contained virtualized scenarios?

Chapter 5

Simulating and testing attacks on drones

Like many cyber-physical systems, drones inherently trust the information provided by their sensors. Furthermore, drones managed by a ground controller also trust the messages sent by this device. This trust is primarily attributed to these systems' real-time constraints, limiting the robustness of any security mechanisms the device may have. This trust and the limited computational resources make drones a prime attack target.

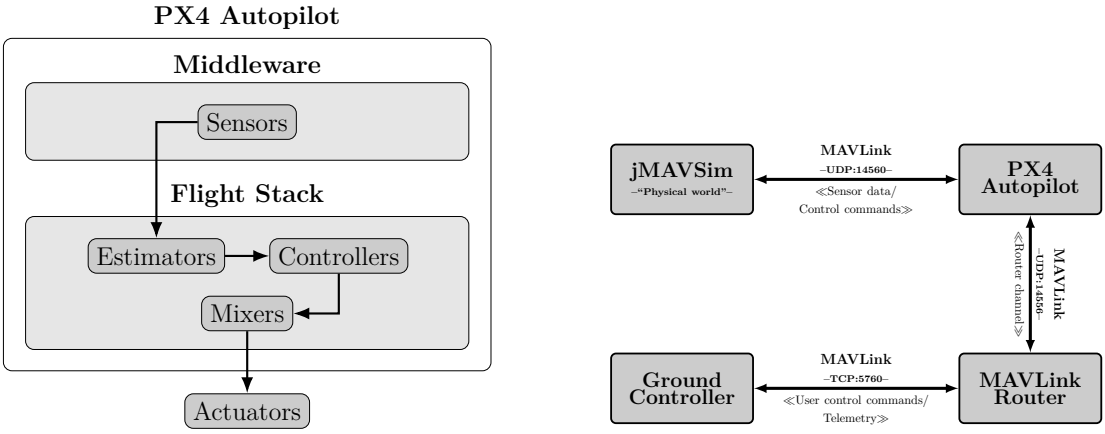
To design and test a defense mechanism against drone attacks, and any cyber-physical system for that matter, accurately simulating a replica of the device and its physical environment provides a way to refine any new implementation without jeopardizing the device.

5.1 Environment description

We conducted all the tests against an Intel Aero Drone hardware running a Dronecore PX4 flight controller. We used the jMAVSim simulator as a drone

replica and a standalone PX4 autopilot as the flight controller to simulate the drone. The drone and the simulator use the MAVLink protocol to communicate with a ground controller. The user can program a flight mission on the drone employing the ground controller. However, while the drone is flying, the user can alter the drone's behavior either with a remote radio controller or by sending specific commands through the ground controller.

A middleware layer and a flight stack comprise the drone's PX4 autopilot firmware (figure 5.1a). The middleware layer communicates with all the device sensors and publishes this information on a message bus [48]. The flight stack then uses this information to estimate and evaluate the commands it needs to send to the actuators.



(a) Overall autopilot architecture **(b)** Communication scheme

Figure 5.1: PX4 autopilot abstraction.

The initial evaluation of the attacks took place inside a simulated environment. In this scenario, the autopilot firmware runs in a host located within a local network. For the sensor data, *jMAVSim* -a multirotor simulator with MAVLink protocol support- was also executed in the same local network. The communication channel between the two components employs the MAVLink protocol (figure 5.1b). A MAVLink router exists between the flight stack and the ground con-

troller in the real drone and the simulation environment as part of the autopilot implementation to translate MAVLink messages from the ground controller's TCP handling to the autopilot's UDP message bus.

To exchange sensor information and rotor output, the autopilot and the simulator send and receive messages encapsulated within the MAVLink protocol [50]. The simulator sends messages with the sensor data (GPS, accelerometer, gyroscope, magnetometer, Etc.); in return, the autopilot sends the actuator control messages. The autopilot and the simulator exchange data asynchronously. The simulator sends each particular sensor message on a specific time-based sample rate for every sensor value. In contrast, the autopilot sends control messages in response to the stimuli received from the simulator.

5.2 Naive drone attacks

The threat model for the attacks relies on a dreadful implementation flaw within the PX4 communication scheme. Since the communications channel employing the MAVLink protocol transmits everything without encryption, MAVLink 2.0 defines an optional signature field as part of the data frame. However, PX4's implementation of the MAVLink protocol ignores this field, and it exchanges all the messages without verifying the origin. Given that the communications channel between a ground controller and a drone does not implement message signing, a third party can send arbitrary messages to the drone over the same communications channel. Since there is no actual authentication mechanism, the drone would acknowledge such messages as legitimate messages.

Moreover, the defacto communication mechanism employed by the drone uses a standard ad-hoc 802.11 wireless network "protected" with a default password that most users neglect to change. This configuration flaw enables an attacker to

connect a malicious host to this access point. An extremely naive attack would entail running the same software used as a ground controller in the malicious host and issuing arbitrary commands directly from the software. Since the protocol has no authentication scheme, an attacker can establish a secondary communications channel with an instance of the same software, and the drone would allow it.

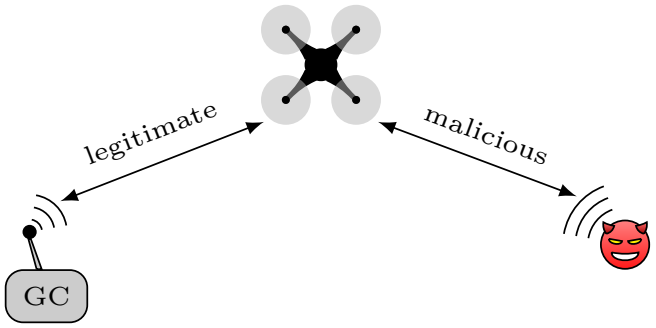


Figure 5.2: Drone attack vector.

The main issue with this type of naive attack is that the legitimate user has the same capabilities as the attacker, allowing the user to counteract most malicious actions sent by the attacker. Due to this fact, extracting some commands sent by the controller and scripting the messages needed to execute a particular action is more efficient. By doing so, the legitimate user might have the same capabilities but not the same speed as a machine sending multiple messages in a specific manner to achieve a particular malicious goal.

To fool the drone into recognizing the malicious host as a ground controller, the latter must periodically send a heartbeat message indicating to the drone that the host is both alive and a ground controller. This deception is achieved by implementing a simple MAVLink communications channel that connects to the drone's MAVLink router and sends the message with ID 0 (a heartbeat).

We captured some specific values needed for communication between the ground controller and the drone while sniffing the legitimate traffic. In particular, the

first heartbeat message has an uninitialized *custom_mode* field and sets the *system_status* field to represent the '`MAV_STATE_UINIT`' state. This initial message is necessary for the drone to register the ground controller. Afterwards, the heartbeat messages are sent with a 'custom_mode' set to `0xc0080600` and the *system_status* set to '`MAV_STATE_ACTIVE`'. Once a host sends both heartbeat messages, the drone acknowledges it as a ground controller and accepts incoming commands from said host.

The first test involves a basic naive attack as a proof of concept, invoking a fundamental action the drone performs. In this case, we chose the "disarm" action, in which the drone effectively shuts down the rotors upon receiving a disarm command issued by the ground controller, regardless of any control logic running on the flight controller. By identifying the message responsible for issuing such a command, an attacker could send the same message during a flight, causing the drone to plummet to the ground.

To capture the 'disarm' command, we initialized a drone with a legitimate ground controller. The rotors were armed, and a 'takeoff' command was issued, followed by a 'land' command and a 'disarm' command. While these actions took place, we captured the messages the drone and ground controller exchanged using a network traffic sniffer. A fundamental finding regarding implementing the MAVLink protocol for these actions is that all messages share the same structure and message ID. All the commands needed to take off, land, arm, and disarm the drone share the same MAVLink message '`COMMAND_LONG`' with varying parameters. This message enables the ground controller to send specific commands with up to seven parameters to the drone. Each command is, according to the definition, identified by an ID. However, the ground controller software implementation uses a custom definition of such commands in which they do not specify an actual

command ID (the value is set to 0), and they specify custom parameter values. In particular, the 'disarm' command sets the parameters 1, 6, and 7 with some custom values to signal the drone to disarm the rotors.

The first naive attack involved sending the 'disarm' command from a malicious host while the drone was in mid-air. We tested this attack on both the simulated environment and the actual drone. In both cases, the drone gracefully accepted the command and plummeted to the ground.

The second not-so-naive attack attempts to force the drone to either change its current course or receive the instruction to go to a particular set of coordinates. To accomplish this, we took a new traffic capture in which we captured the command to send the instruction to travel to a specific latitude, longitude, and altitude. The attack entails repeatedly sending a message to the drone, instructing it to a specific location. Afterward, the malicious host monitors the telemetry information broadcast by the drone to all registered ground stations to determine whether the drone is reducing its distance from the desired destination. If a significant increase in distance is detected, the automated attack script sends the same instruction again, forcing the drone to resume the malicious trajectory. Once the drone is reasonably close to the target location, the script sends the drone a 'land' command.

While all the naive attacks were successful, the solution for this attack vector is relatively simple: implement the message signature scheme defined by the MAVLink protocol version 2.0. This solution would prevent an attacker from sending arbitrary messages to the drone without having the necessary key to sign the messages while avoiding using encryption schemes over the entire communications channel.

5.3 GPS attacks

Tampering with GPS signals is a complex and unreasonable technique in research, as it will affect the desired target and any undesired nearby devices that rely on this technology, potentially affecting unaware users of these devices. This fact places an ethical constraint on any attack tests involving GPS signals.

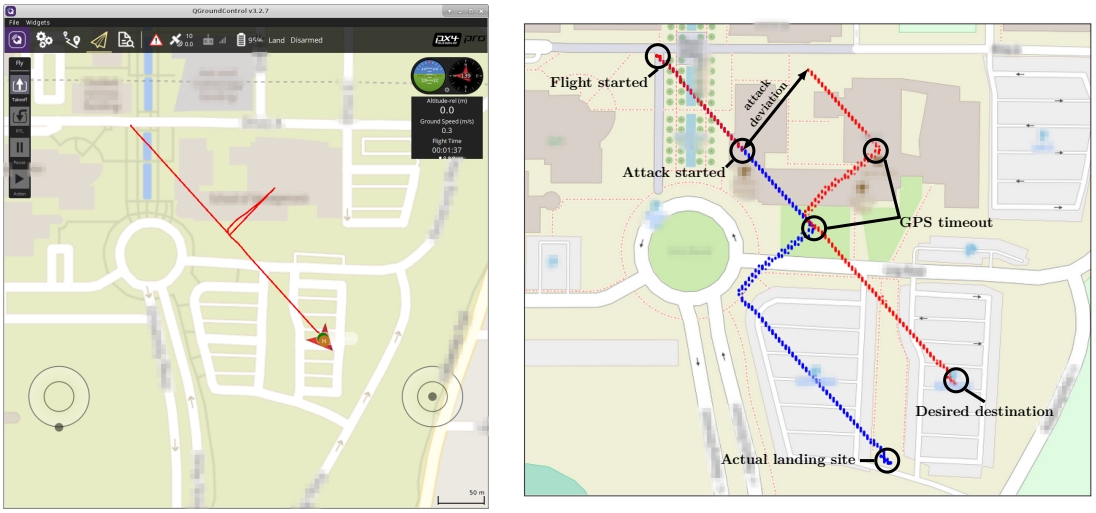
To simulate the tampering of the GPS sensor data, we took advantage of the scheme used by the simulator to send the sensor data to the PX4. As depicted in figure 5.1b, the sensor data is encapsulated inside MAVLink messages exchanged between jMAVSim and the PX4 autopilot. We can tamper with sensor data by intercepting these messages before arriving at the autopilot's middleware.

We identified that the message carrying the GPS coordinates is the MAVLink message with the ID 113 (`HIL_GPS`), which must be intercepted and modified to tamper with the GPS Coordinates. This message carries the current GPS coordinates simulated by jMAVSim, effectively being the GPS coordinates provided by a regular GPS device within an actual drone. By modifying these coordinates, an attacker can create a deviation between the exact physical location of the drone and the location received by the autopilot. This deviation will, in turn, force the autopilot to estimate its calculations based on false readings, compelling the drone to behave unexpectedly throughout the attack.

Inside each `HIL_GPS` MAVLink message, both latitude and longitude are encapsulated as two integers, representing the degrees x10^7. For instance, a latitude of $1°0'0''$ is the integer 10,000,000. Since each degree is equivalent to roughly 111 kilometers (69 miles), by adding 5,000 to any GPS coordinate value inside a MAVLink message, that tampering would create a deviation of approximately 55 meters (60 yards). We consider this deviation a reasonable one for testing purposes.

Item	Value
GLOBAL_POSITION_INT.lat	3.298559450000000e+8
GLOBAL_POSITION_INT.lon	-9.674850060000000e+8
GLOBAL_POSITION_INT.alt	1.901530000000000e+5
HIGHRES_IMU.xacc	3.301204741001129e-2
HIGHRES_IMU.yacc	-1.333707012236118e-3
HIGHRES_IMU.zacc	-9.831379890441895e+0
HIGHRES_IMU.xgyro	-2.466390375047922e-3
HIGHRES_IMU.ygyro	1.280557457357645e-2
HIGHRES_IMU.zgyro	5.454805679619312e-3
HIGHRES_IMU.xmag	8.958292603492737e-1
HIGHRES_IMU.ymag	2.638268284499645e-2
HIGHRES_IMU.zmag	1.798238515853882e+0

Table 5.1: Sample sensor data extracted from a HIL_GPS message.



(a) Drone trajectory observed by the user.

(b) Legitimate (blue) and tampered (red) sensor data.

Figure 5.3: Drone's GPS deviation.

Whenever we apply such a deviation during a routine flight between two coordinates, the user will observe a sudden "jump" that the drone makes throughout its trajectory. Once this jump occurs, the autopilot starts correcting its course accordingly to complete its current task. The user observes this correction as a different trajectory of the drone returning to its original path and then resuming

its flight (figure 5.3a).

Three stages comprise this GPS attack. While the drone is flying, performing its routine tasks to achieve its goal, we start the attack. At this stage, the drone discards the deviations created by the tampering as merely GPS imprecision. As time passes and a GPS timeout occurs, the autopilot begins to acknowledge the new reported position as the actual physical location of the drone. At this point, a second stage of the attack begins. The autopilot starts compensating for the deviation introduced by the attack by "returning" to its original trajectory. However, this compensation is a deviation from the original path in the opposite direction of the deviated coordinates. Once the autopilot makes the corrections, the final stage in which the drone resumes its flight within a corrupted trajectory begins, resulting in the drone reaching a tampered destination while reporting the arrival to the original destination.

Figure 5.3b depicts the entire attack, which shows both the data received by the attacker (the sensor data) and the data sent to the autopilot (tampered data). The former is drawn in blue, while the latter is in red. The figure shows how the autopilot reacts to the tampered coordinates. At first, a timeout must occur to acknowledge the anomalous sensor data as legitimate. Once that timeout occurs, the flight controller initiates a trajectory correction routine to return to the original path.

5.4 Simulation feasibility

In both instances, we examined how simulated attacks can seamlessly translate to actionable attacks against the target drone. These two types of attack are based on the premise that the attacker can either freely communicate with the drone's autopilot posing as a ground controller, or they can tamper with the sensor

readings and reporting.

Simulated environments provide a crucial proving ground for cybersecurity research because they allow for the exploration of complex attack scenarios without risking actual harm to the system or the surrounding environment. By using simulations, researchers can meticulously dissect the attack vectors that could compromise a CPS, understanding not just the immediate effects on the targeted system, but the potential cascading effects throughout the interconnected network. This approach is particularly beneficial in comprehending the consequences of attacks on systems where real-world testing would be impractical or ethically unacceptable.

Furthermore, simulations afford the unique advantage of repeatability under controlled conditions. Researchers can iteratively adjust the parameters of an attack, pinpointing the precise vulnerabilities within the system's architecture or its operational protocols.

The fidelity of these simulated environments has increased dramatically, allowing them to accurately mirror the complexities of real-world cyber-physical systems. This fidelity ensures that the insights gained are highly applicable and can significantly enhance the security posture of CPS against a wide array of cyber threats.

In essence, the utilization of simulations as a testing ground for cybersecurity research on generic cyber-physical systems is not only feasible but imperative. This methodology enables a proactive approach to security, where potential vulnerabilities can be identified and addressed before they are exploited in the real world, thereby safeguarding the integrity and functionality of critical cyber-physical infrastructures. Moreover, they allow the researchers to conduct these tests without compromising the physical integrity of the system.

Chapter 6

Using Software-Defined Networks to secure Industrial Control Networks

As part of our approach to creating a simulation framework for cyber-physical systems, we evaluate the advantages of incorporating software-defined networking. Since we already observed the feasibility of testing attacks in a simulated environment, we now focus on defense mechanisms.

Most of the literature on Industrial Control Systems (ICS) security has focused on *preventing* and *detecting* attacks [19]; however, *responding* to attacks has received much less attention [19, 2]. In particular, most research papers focusing on intrusion detection for control systems do not discuss what to do after an attack has been detected [20].

To address this gap, we propose an intrusion-response architecture that leverages Software-Defined Networking (SDN) to automatically reconfigure the attacked network in real-time, based on alerts generated by an IDS. While previous

attempts focused on replacing compromised sensor and control messages with trustworthy sources [45], we focus on two new use cases for network reconfiguration during an attack:

- We show how to create a honeypot of an industrial system to reroute the attacker to this honeypot.
- We show and work around the challenges of creating a honeypot when the attacker is already inside the industrial control network.

For our purposes, we need to implement a way to give an external system a copy of all packets in the managed network. Since we would like to manipulate the network dynamically based on the current state of the simulated system, we need to implement a mechanism that can control the network's behavior via the northbound API.

Since grouping allows the switch to send packets through multiple ports, we can add an entry to redirect a copy of all the traffic to a single passive interface attached to the external system. Group entries are then defined to acquire a copy of each packet sent through the corresponding port of each device connected to the SDN switch toward this system. This effectively obtains a complete copy of the network packets within the protected network.

6.1 System description

Figure 6.1 shows the three main components that comprise the ICS: a physical process, a field network (a network between field devices and controllers), and a supervisory network (a network between controllers and SCADA systems). A secure architecture for ICS [43] suggests that these components should be placed in layers so that, ideally, an element within a given layer can only communicate

with other elements of the adjacent layers when necessary. Other interactions between layers are possible but usually discouraged.

Since the controller is more of an SDN than an ICS component, it must be placed accordingly. We conceive this element as the management component of all the network requirements of both the field and supervisory networks. As such, it must interact, if needed, with SDN elements within these networks.

However, due to security concerns and the essential role this element will exert over the scenario, direct communication between the components involved in the actual physical process and the SDN control network is not desired. Considering this constraint, a new layer above the field and the supervisory networks within the ICS architecture contains the SDN controller.

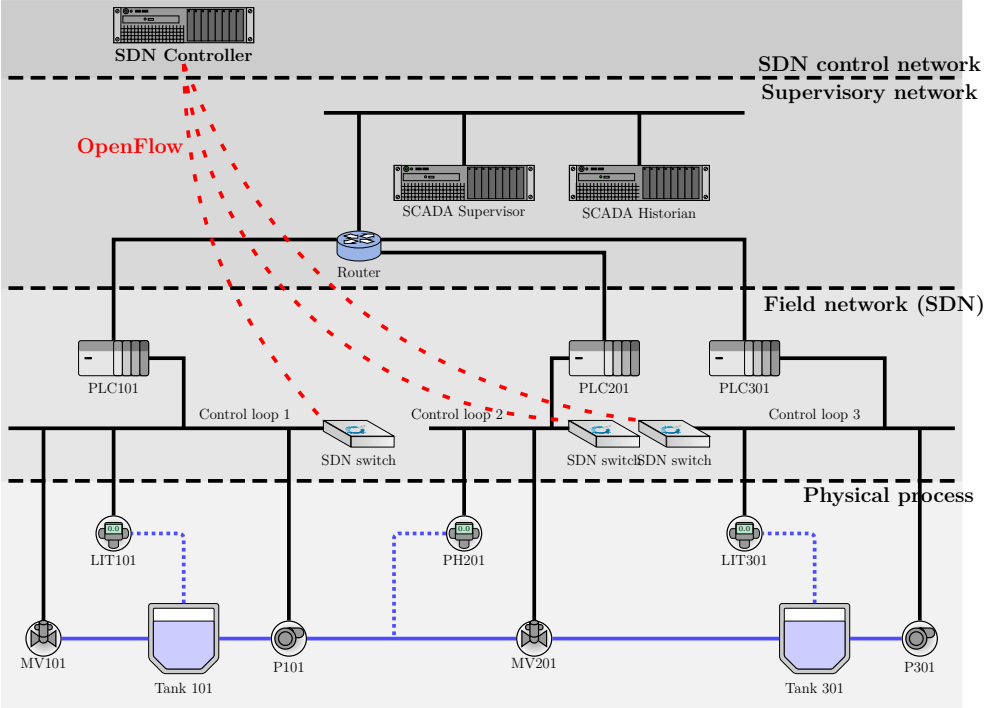


Figure 6.1: ICS network layers with SDN.

Figure 6.1 depicts the scenario used as a general testbed for our incident response scheme. In this scenario, three PLCs control a simulated physical process

representing a water purification plant. An SDN virtual switch is managed by an SDN controller and handles the data plane of all the communications in the network.

We assume the attacker has compromised the field network. They achieve this compromise either externally or internally (Figure 6.2). How they acquired the privileges required to gain access to the network is out of the scope of this scenario. We assume that they already executed such actions and gained the necessary privileges.

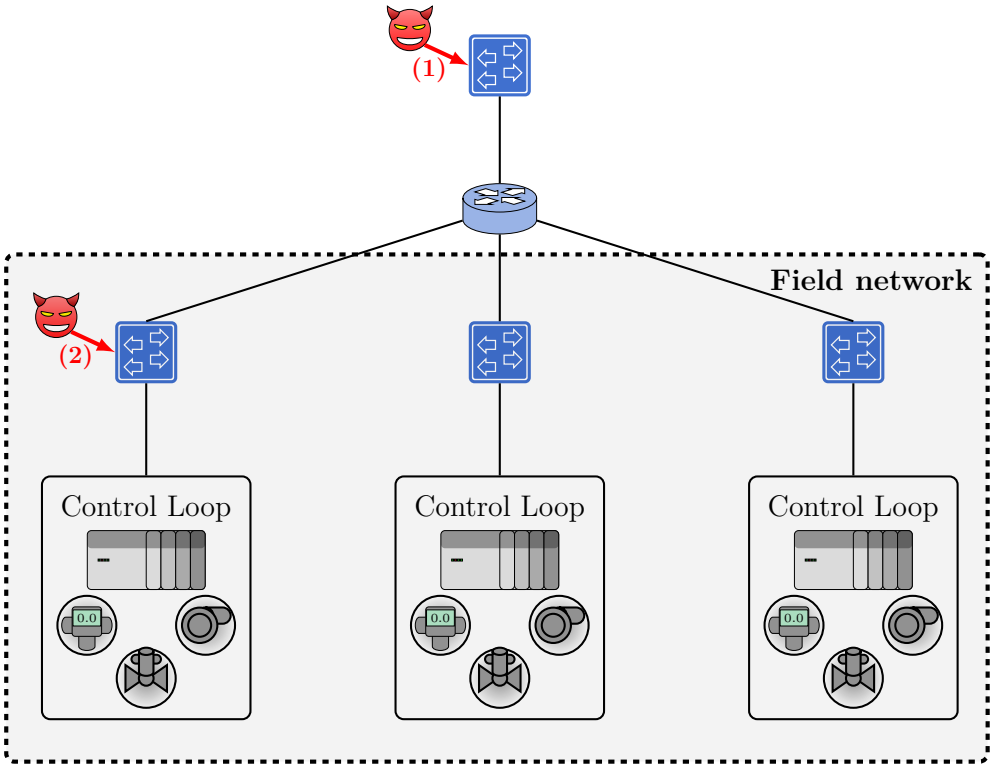


Figure 6.2: Theat model against the ICS.

The threat model includes two different kinds of attackers. The first attacker contemplated by the model is an external attacker who gains access to the field network (1). The second is an attacker located directly within the field network (2). Regardless of the kind, they can communicate with the devices inside each

control loop directly or through some routing. In other words, they have a communications channel between the attacker's location in the network and the target devices.

6.2 Intrusion response scheme

Figure 6.3 depicts our design for an external IDS. The main idea behind this scheme is to obtain a copy of all the network traffic flowing within the field network and relay that traffic to an external IDS. This system is implemented within the same SDN control network, as it must communicate with the controller to alter the SDN due to an alert dynamically.

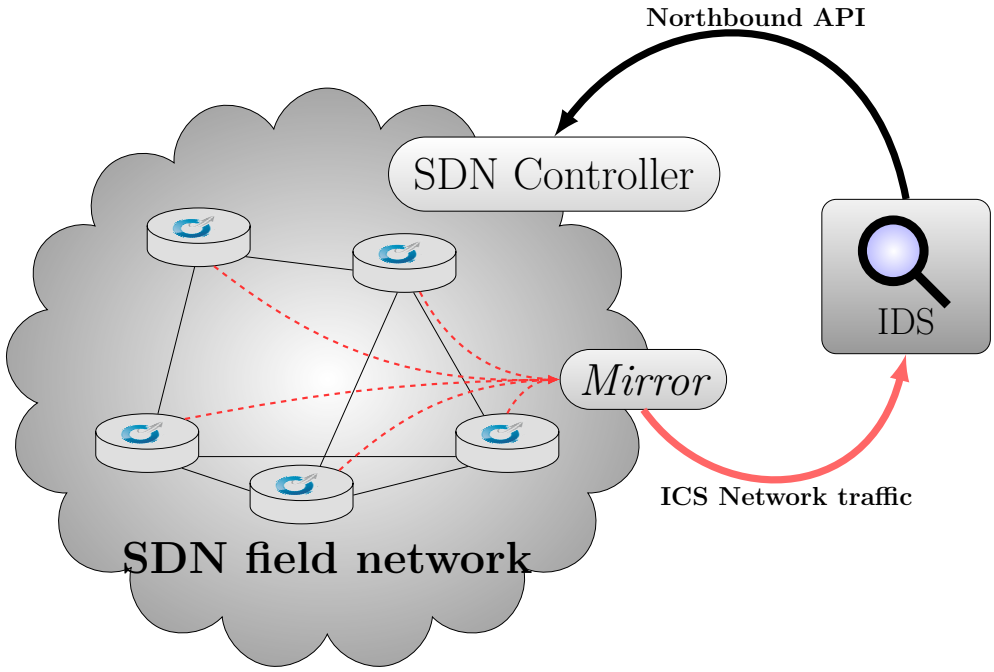


Figure 6.3: Intrusion detection scheme.

The intrusion detection scheme uses mirrored traffic acquired from all the OpenFlow devices the controller manages. Upon detection, the IDS reacts to defined attacks by sending northbound API commands to the SDN controller to

prevent further attacks.

An external IDS can detect anomalous behavior of the physical process itself. Furthermore, it can also analyze the network traffic with traditional network IDS to identify known malicious techniques within the field network. Moreover, any commands sent from the IDS to the SDN controller would not be accessible from within the field network, preventing the attacker from realizing any countermeasures the IDS took.

Some related work has already addressed some challenges involving using an IDS with SDN regarding the data capture [71]. Despite the many benefits that the SDN provides, the segregation of the control and data planes makes it challenging to inspect the complete packets circulating within an SDN. As a possible solution, Yoon et al. [71] present a conceptual design in which the application layer that handles the northbound API of a controller acquires the traffic from the SDN network through a separate NIC in an in-line fashion. However, they mention the possible performance issues this might pose for the SDN as a whole. They also provide some insights regarding the implementation with passive modes, in which the controller requires additional network interfaces between the control and data planes to collect the entire payload, which is impossible via the control channel.

For our purposes, we consider that a passive-mode implementation is adequate for the intended solution. Not only does this implementation allow the inspection of the packets, but it also avoids affecting the overall performance of the SDN controller. Therefore, our scheme uses a passive-mode IDS in a separate machine, communicating with the SDN controller via the controller's northbound API.

6.2.1 ICS Honeypot

The main idea of our defense mechanism is to take advantage of the SDN controller to execute the necessary actions to relocate an attacker detected within the field network into a honeypot network. This honeypot network must contain simulated replicas of the physical devices implemented in each control loop, creating the illusion of an actual working ICS with all its inner workings and communications between the components.

We used miniCPS to implement the honeypot because it provides valuable features to simulate a pre-defined ICS on top of a virtualized SDN using mininet. At this stage, our work focuses on the actual usage of SDN technology rather than the simulation of the devices themselves, which we will address in the future. We point out that the previous efforts already partially tackled the simulation of the process, which includes the implementation of the EtherNet/IP stack as the communications protocol for the simulated ICS.

There are four components to the defense scheme: the SDN controller (ONOS), the field and honeypot networks (Mininet and MiniCPS), and the IDS. ONOS will control the networking, manage the interactions between the field network and the honeypot, and the mirroring scheme to deliver the network traffic to the IDS. The latter, in turn, will receive the raw network packets from both the field and honeypot networks to make the appropriate security decisions, which are then translated to REST API commands that are sent to the controller as needed (figure 6.4).

In addition to these external components, the field and honeypot networks need to communicate whenever needed. However, only specific cases where the field devices might be jeopardized should allow this communication. In other words, there has to be a bridge between both networks that should only allow

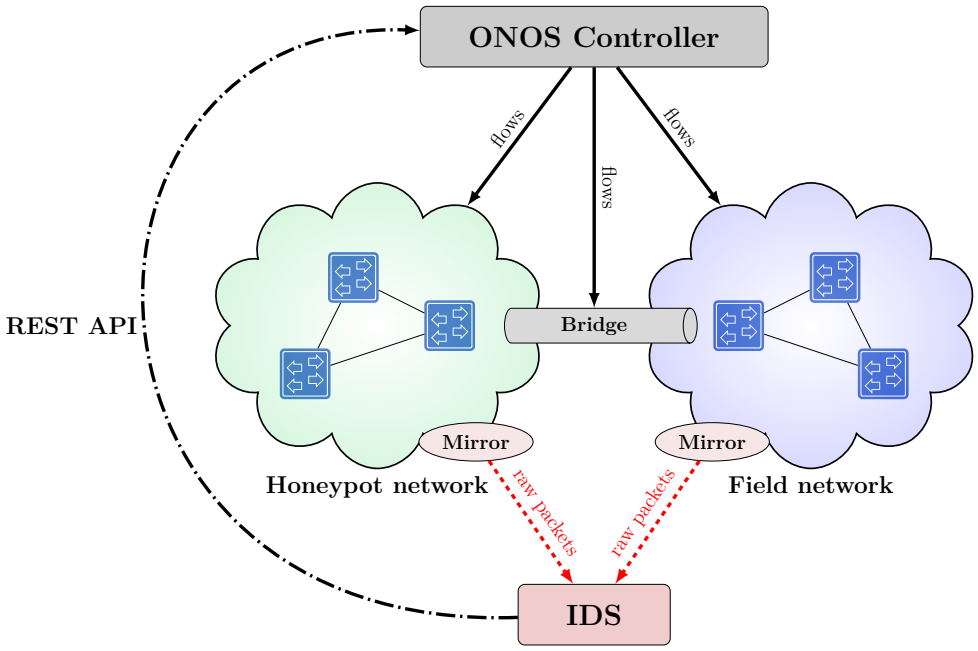


Figure 6.4: Fundamental interactions between the components of the ICS defense scheme.

traffic to flow upon specific requests by the IDS. Under normal circumstances, this bridge should effectively insulate both networks.

While probably the most intuitive method of implementing this bridge would be establishing a link between each pair of SDN switches in each control loop between the field network and the honeypot, this approach could depend on the router to forward traffic between different control loops within the honeypot. This forwarding would mean that the router must know the honeypot when the whole idea is to be invisible from the existing network. Therefore, having a centralized bridge allows the field and honeypot networks to communicate as needed without depending on the router. Every routing decision can be defined via flow table entries as required, while regular communications between these networks can be restricted by default, making the networks invisible to each other.

Whenever an attacker is identified and isolated inside the field network, the IDS effectively relocates the attacker within the honeypot. We send the appro-

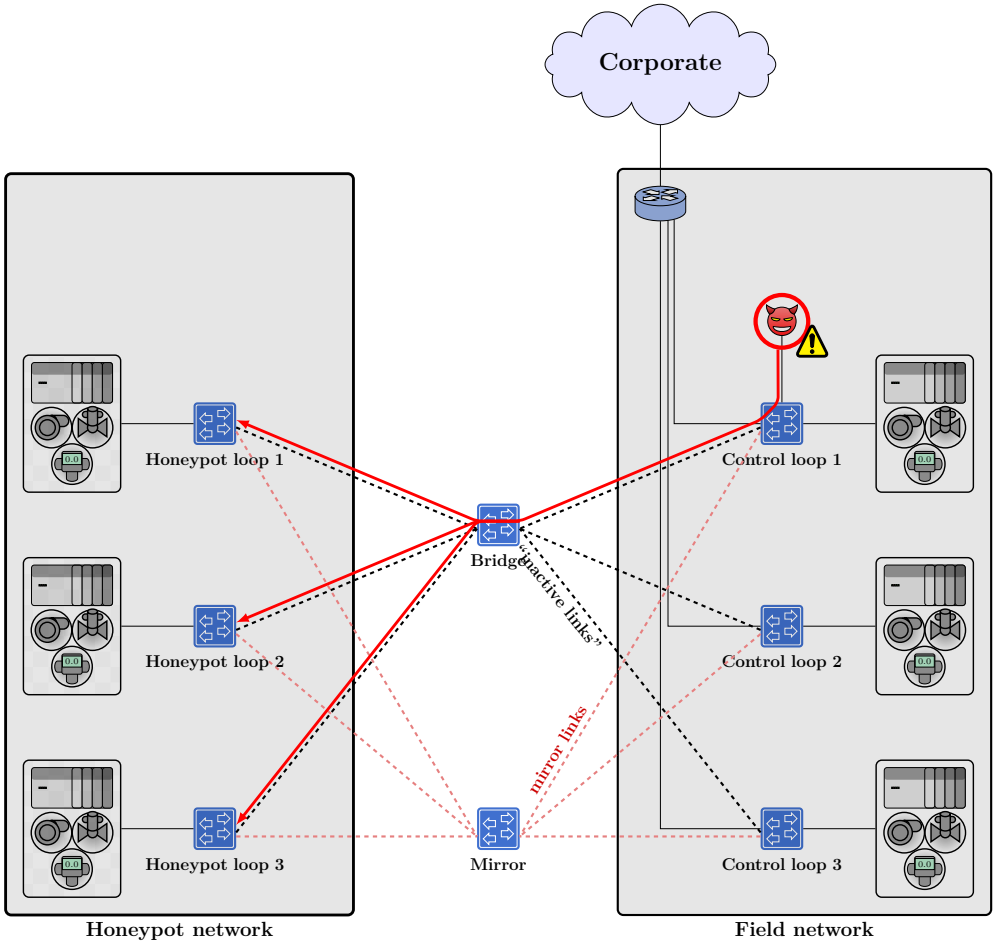


Figure 6.5: Attacker relocation.

priate instructions to ONOS from the IDS via the REST API. These instructions include the flow table entries that allow the SDN switches to reroute all the traffic incoming from the attacker through the bridge and into the honeypot, as well as the corresponding replies from the fake devices within the honeypot back to the attacker (figure 6.5). Under normal circumstances, flow tables in the bridge switch prevent traffic from flowing between the field network and the honeypot, hence the name "inactive" links. Upon detection of an attacker, flow tables are modified as necessary to reroute all the traffic involving the attacker into the honeypot.

6.3 Performance tests

Since this scheme is meant to be a seamless transition between the existing network and the honeypot network, it is paramount that attackers do not realize that they have been moved to the honeypot network. This means that not only the real scenario must be replicated, but the behavior of the network must be maintained.

To accomplish a seamless transition from the existing network to the honeypot, the SDN controller must relocate all the traffic sent by the attacker and redirect it to the honeypot so that the activities performed by the attacker appear to be continuous. The success of this approach will confirm whether SDN is a suitable mechanism for our future endeavors regarding the simulation framework.

We performed two main tests in the scenario depicted in figure 6.6. The first one involves a custom UDP service. We chose a UDP service because we need to measure the response time of the traffic itself while transitioning between the two networks. Moreover, a connection-oriented service could change the test outcome due to possible re-connection times. Therefore, a service that does not require a connection will indicate the actual response times of the network as a whole. This connection-less nature means that the necessary flow table updates on the SDN devices along the path will account for any delays in the communication.

The custom UDP service is a simple service that replies 8 bytes of data to any arriving packet. The 8 bytes are just a timestamp coded in a long integer. Whenever the client sends a packet, it records its local timestamp and calculates the difference between the timestamp it sends and the client's timestamp by the time it receives the response. Note that the calculation only uses the local timestamp. The remote timestamp is only used as a payload for the packet, meant to add a standard-sized payload to the traffic between the client and service. This

difference in time will be the service's round-trip time (RTT), which is the time it takes to send a packet, be "processed" by the service, and receive a response.

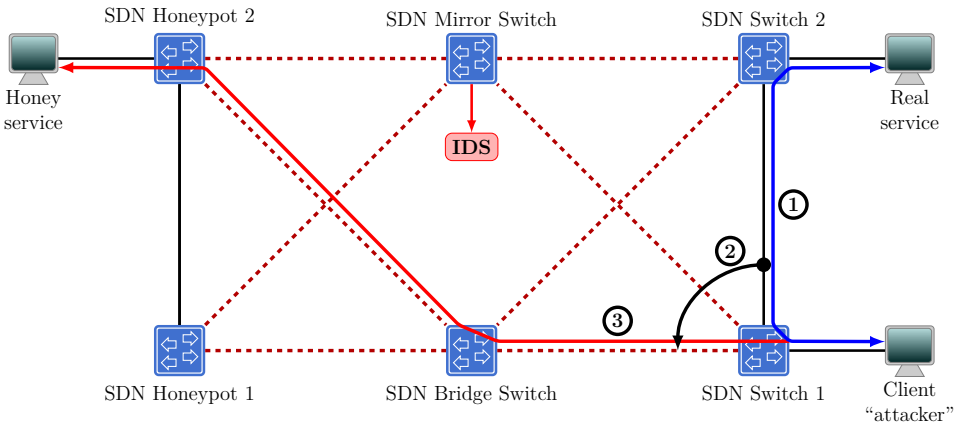


Figure 6.6: Performance test scenario.

Figure 6.6 shows the performance test scenario. Two machines execute the same service both in the field and honeypot networks. An "attacker" located in the field network will initially exchange data with the genuine service (1). Upon executing the relocation mechanism, the SDN controller reconfigures the switch flow tables to redirect the traffic to the honeypot network (2). The attacker then exchanges data with the honeypot service (3). To get a sense of any noticeable delay, the attacker logs the RTT of each packet sent to the service, unaware of the service's location throughout the process.

While the test runs, we configure a set of flow table rules in the SDN controller, forcing the switches to redirect the client's traffic ("attacker") to the honeypot network. Since all the packets will have the wrong Ethernet frames, the attacker must request the "new" MAC address to continue using the service. Furthermore, the SDN devices that have just updated their flow tables will add the rules whenever traffic matches every new rule. Meanwhile, since the packets do not reach the intended destination, the attacker experiences a timeout in the communication.

The result of this redirection, as seen by the SDN controller, is depicted in

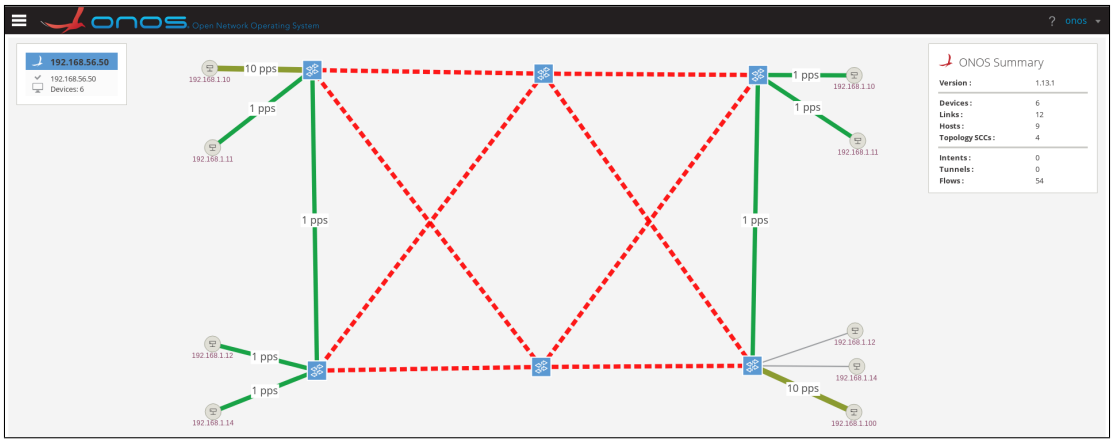


Figure 6.7: An execution of the test as seen by the SDN controller.

figure 6.7. All the traffic generated by the host with IP address 192.168.1.100 -located in the field network on the right- is sent to the honeypot network on the left. In this case, the host is contacting the service published by the IP address 192.168.1.10, which is exposed in both hosts sharing that IP address in the field and honeypot networks, making a seamless transition possible. The remaining hosts in both networks communicate with each other within the boundaries of each network. Notice that the links between both networks appear as dashed red lines, indicating a "disabled" or "disconnected" link. This disconnection results from a set of flow table entries used to isolate both networks.

Figure 6.8 shows the collected results of a sample execution. For each packet sent, the attacker logs the RTT of that particular packet with the current timestamp. This data can depict the behavior of the SDN network while transporting the data between two hosts. The gap highlighted in grey represents the downtime experienced due to the relocation of the attacker from the field network to the honeypot via flow tables. As with any other regular traffic in an SDN, the first packet of a new connection has a larger RTT than the remaining traffic associated with a particular communications channel between two hosts.

To estimate the behavior of the SDN to such relocation, we performed this

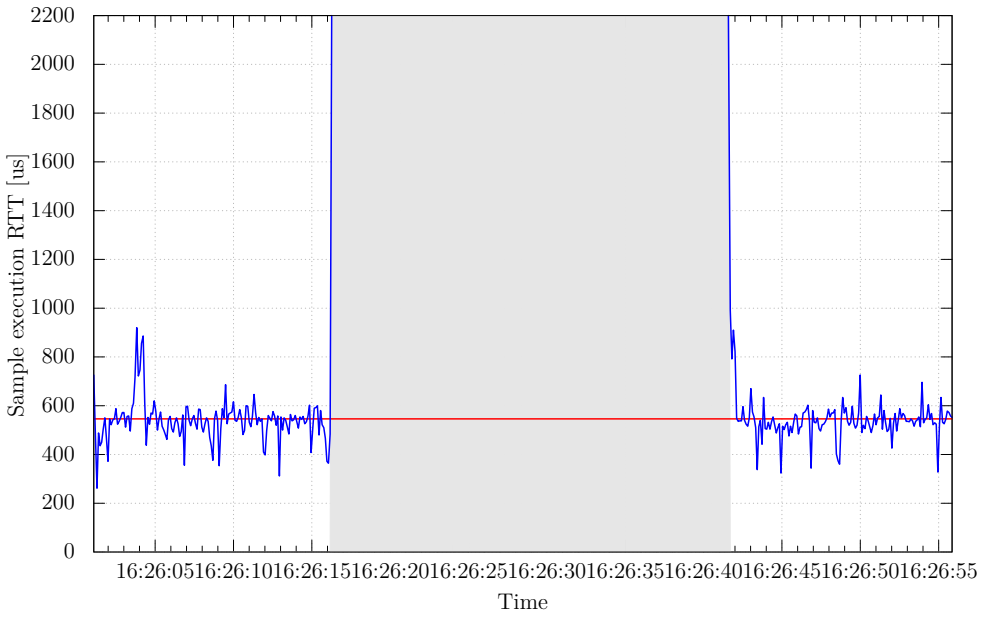


Figure 6.8: An execution of the performance test.

Total packets	314
Received packets	291
Lost packets	23
Packet loss	7.32%
Execution time [s]	54.774531
Connection uptime [s]	30.544737
Connection downtime [s]	24.229794 (44.24%)
Average RTT [us]	546.05
Recovery RTT peak [us]	2,034

Table 6.1: Metrics of an execution of the performance test.

test 100 times. Each time, the attacker logged the RTT of every sent packet and either the time it took to get a response or a timeout of 1 second. We performed each execution in the following manner: the attacker sent data for 15 seconds, then the relocation took place, and finally, an additional 15 seconds of data transmission. The idea is to have a standard 30-second uptime to reach the "intended" destination for the payloads. The intended destination would be the genuine service; in the honeypot, the fake service.

Since the main challenge of the relocation is to avoid the detection of the honeypot by the attacker, the relocation timeout (i.e., the time it takes to receive a response after the relocation takes place in the controller) is a variable that must be considered as part of the performance of the protection scheme, this delay in the communication must be slight enough to avoid any concerns by the attacker. Figure 6.9 presents the measured downtime on every test execution. The delay experienced by the client service appears to have a random component. The changes in the delay revolve around 25 seconds, which means that this is how long the attacker will experience a service outage.

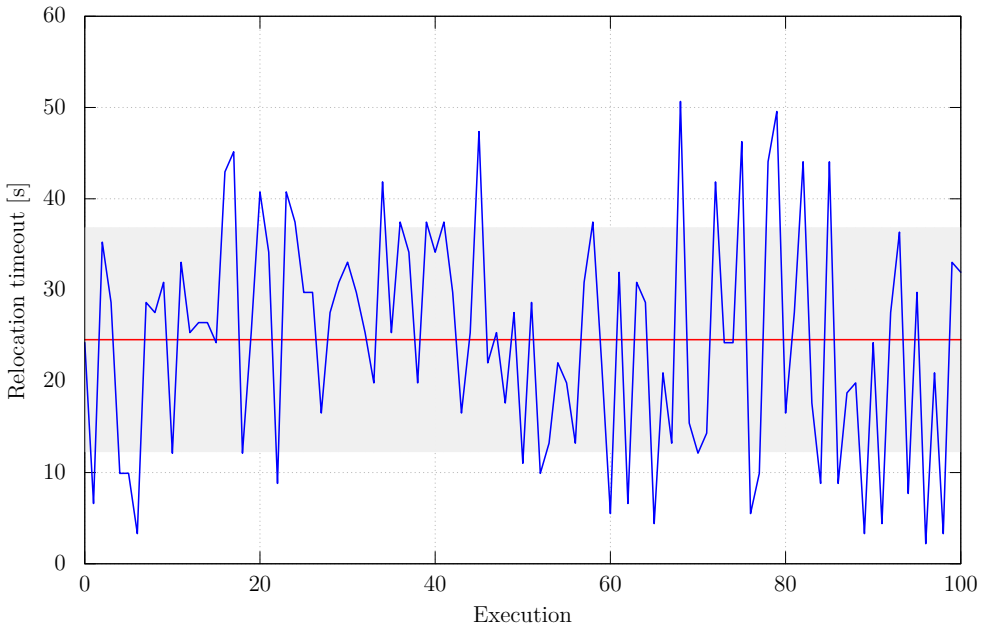


Figure 6.9: Downtime measured on each execution of the test.

To fix this issue, we look at the origin of this delay. As it turned out, it was caused by how the relocation process generates the flow table entries. At this point, the flow entries match the attacker's address, and as an instruction, the entries have an OUTPUT action towards the port connected to the bridge between the field network and the honeypot network. While this effectively redirects all the

attacker's traffic toward the honeypot, there is an issue with the MAC addresses. The network should deliver all the traffic that the attacker is sending to the original device. While the honeypot device has the same IP address, the MAC address of the emulated device is different from the original one.

A naive solution would simulate the honeypot devices with the same MAC addresses. However, as the internal indexing of hosts within the SDN controller is MAC addresses-based, this would be counter-productive as legitimate traffic would be sent to the honeypot, and innocuous traffic from the honeypot would be sent to the field network, ultimately affecting the physical process.

We added new flow entries in the honeypot switches to solve this issue. The idea behind these new flow entries is that they allow us to modify the packets to change the destination MAC addresses. The solution is to change the MAC address to the corresponding addresses of the honeypot device whenever an attacker's packet is rerouted; the same thing must be done in the opposite direction, and every packet originating from the simulated device must be altered to change the source MAC address to that of the original device. The requests for the service will be redirected to the honeypot service, and the responses will be correctly routed to the attacker. After implementing these changes, figure 6.10 shows the RTT of a sample execution under these conditions. Changing the MAC addresses significantly improved the response time to the point that, at most, a single packet is lost.

When done correctly, looking at these results, the relocation mechanism is reasonable for identifying and removing a detected attacker from a network. If a convincing honeypot is deployed in the dummy network, the attacker can continue to "interact" with the system without realizing it is a honeypot.

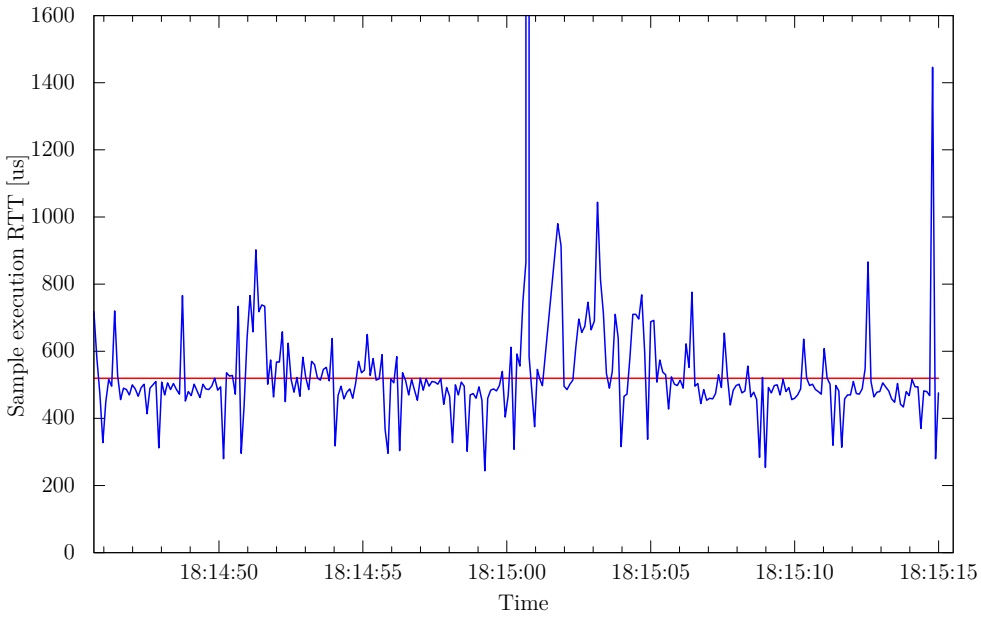


Figure 6.10: Another test execution after the MAC address issue was resolved.

Total packets	302
Received packets	301
Lost packets	1
Packet loss	0.33%
Execution time [s]	30.831129
Connection uptime [s]	30.803247
Connection downtime [s]	0.027882 (0.09%)
Average RTT [us]	531.57
Recovery RTT peak [us]	1,057

Table 6.2: Metrics of executing the performance test after the MAC address issue was resolved.

6.4 Simulating with Software-Defined Networks

The test scenario we used, in which we tested attacks and defense schemes against an ICS, highlighted the advantages of using SDN features as part of a complex simulation. Moreover, its flexibility allowed us to test a custom environment, simulating a cyber-physical system. While the simulation was built by a third party, these tests allowed us to corroborate the usefulness of SDN for this

endeavor.

Using an SDN as a base for the simulation, we can dynamically generate custom networking scenarios and incorporate various simulations to mimic several complete systems. This opens the door to developing a custom solution that can aid us in our goal to create a research framework capable of emulating a reasonably realistic version of a system.

Chapter 7

Network Emulation Framework for Industrial Control Systems

The primary motivation for developing this framework is the current situation between Russia and Ukraine. In less than a decade, Ukraine has suffered from several cyber attacks attempting to cause electrical outages. On December 23, 2015, in a dark and cold winter, Ukraine suffered a blackout caused by cyber attacks [74]. This was the first confirmed case worldwide of cyberattacks intentionally targeting the power grid. In this first incident, attackers gained remote access to the industrial networks of power companies, and a remote adversary operated the human-machine interface of operators, opening circuit breakers manually.

A year later, on December 17, 2016, a fifth of Ukraine's capital, Kyiv, experienced another blackout [22, 46]. This time, the target was a transmission utility, and unlike the previous year when remote human attackers opened the circuit breakers, the attack in 2016 was launched automatically by the first known industrial malware targeting the power grid: Industroyer [8] (also known as CrashOverride [11]).

Finally, on April 8, 2022, operators discovered another malware tailored to

attack circuit breakers automatically in the first months of the Russian invasion of Ukraine. This new malware was called Industroyer 2, and it represented yet another attempt to target Ukraine's power grid amid many other Russian cyber-attacks coordinated alongside their kinetic military action [35].

The two Industroyer malware payloads represent a watershed moment for the security of critical infrastructures for several reasons: (1) they represent the first and only known malware samples specifically deployed by malicious actors to target the power grid. (2) Industroyer is the only known malware that caused a power blackout successfully. (3) Industroyer 2 exemplifies how modern wars combine cyber and kinetic attacks to maximize a show of force. (4) While the attackers in 2015 needed a real-time network connection to the power company to open circuit breakers, Industroyer and Industroyer 2 can, in principle, be deployed in air-gapped systems (or systems with robust network segmentation where remote connections are not allowed) because the malware could automatically target power grid equipment without the need for human feedback. (5) Industrial malware may have other advantages over remote (manual) attackers, like the ability to repeatedly send commands to devices faster than an operator could respond.

In the remainder of the document, we will refer to "Industroyer" as "*Industroyer 1*" to avoid ambiguity.

We believe academic researchers working on the security of the power grid want to learn the details of how Industroyer 1 and 2 launched the attacks and the impact that similar malware may have in the future so that we can understand the attacker's decision-making behind the payload design and also anticipate future attacks. However, as stated before, any tests or execution of these artifacts in actual power grid systems is unfeasible and dangerous. Thus, a secure virtual environment is necessary.

7.1 Framework design

The typical design to simulate physical systems involves translating all the different physical components into some non-linear system described by several differential equations. After doing so, the simulation follows these equations to reflect the system's physical state. As with every solution, this approach has both advantages and disadvantages. On the one hand, the simulator has complete knowledge of every variable in the system, as the equations characterize the system entirely. Moreover, as the simulator holds all the variables, any response is given without relevant delay. On the other hand, the simulation is fixed to a specific scenario and has no flexibility to scale up without a significant update to the system's equation model. Moreover, the model describing the system being fixed means that all the information must be processed in the same simulation process, regardless of how many threads the simulator uses, which means a single end-point simulating the whole system.

Considering these advantages and disadvantages, we decided that our simulation should break apart the model into simpler mini sub-models that can work autonomously, regardless of the scenario's topology. By building the simulator, we mean determining the simulation model of a single RTU and running it as a stand-alone simulator in a single end-point. This way, we do not need to know the topology beforehand, and the simulator can accommodate different physical topologies without changing the underlying model. However, the disadvantage comes with an added delay, as we need some mechanism for the separate RTUs to exchange messages regarding their current state.

The framework design has three main components: Physical processes, network protocols, and network infrastructure. Employing this design pattern allows the future deployment of additional modules to support a diverse set of protocols

and processes.

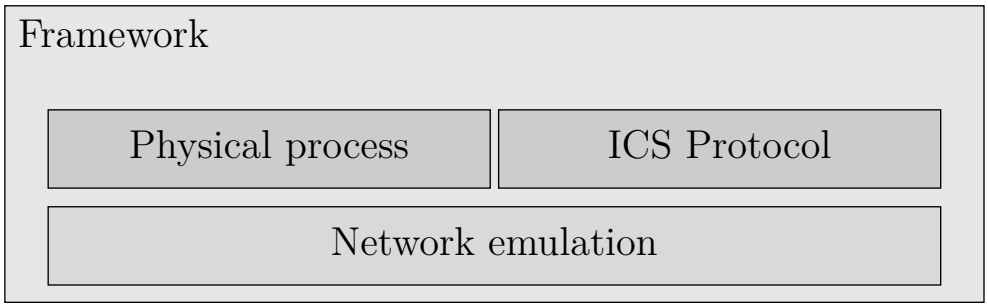


Figure 7.1: Framework modular design.

Based on this overall design and the requirements we identified in the dissertation scope, Figure 7.2 illustrates the proposed design. **Isolation**, **Secure-coupling**, and **Network traffic** are three characteristics that a virtual network linked to a virtual machine's network interface offer. Two virtual machines linked by a host-only network in the hypervisor prevent potentially hazardous network traffic from leaving the secure environment (Figure 7.2 ②). Moreover, having all the malicious traffic routed to the simulated network via a virtual machine interface allows us to capture any network traffic for analysis.

To implement the virtual network itself, kernel namespaces, often used in container solutions, provide a viable way of simulating multiple hosts in a single Linux machine. Mininet uses the same concept to create virtual network hosts. Since it is Python-based, we can develop a way to integrate a physical simulation into the network emulation provided by Mininet. Moreover, Mininet uses Open vSwitch to interconnect the simulated hosts, providing a software-defined network solution that allows us to connect the simulation to a virtual machine's host-only network interface, integrating the infected host to the simulated network (Figure 7.2 ①). This solution offers a secure and isolated network to capture and analyze the network traffic.

We can achieve **Flexibility**, **Extensibility**, and **Customizable** characteris-

tics by implementing the sandbox in separate modules. Any industrial control protocol supported by the sandbox would be a separate module, dynamically loaded and enabled as required. By making the sandbox modular by design, we can incrementally add new physical processes and industrial control protocols as needed. Moreover, implementing a handler component to parse a configuration file with the description of the desired scenario allows us to build complex topologies dynamically without explicitly developing the scenario but rather the modules comprising it.

Finally, the **Physical emulation** must provide realistic data over the implemented protocols. We must interconnect the physical simulation with the network simulation (Figure 7.2 ③). For this purpose, we identified two primary components for our sandbox: the communication with the network emulation and the physical process.

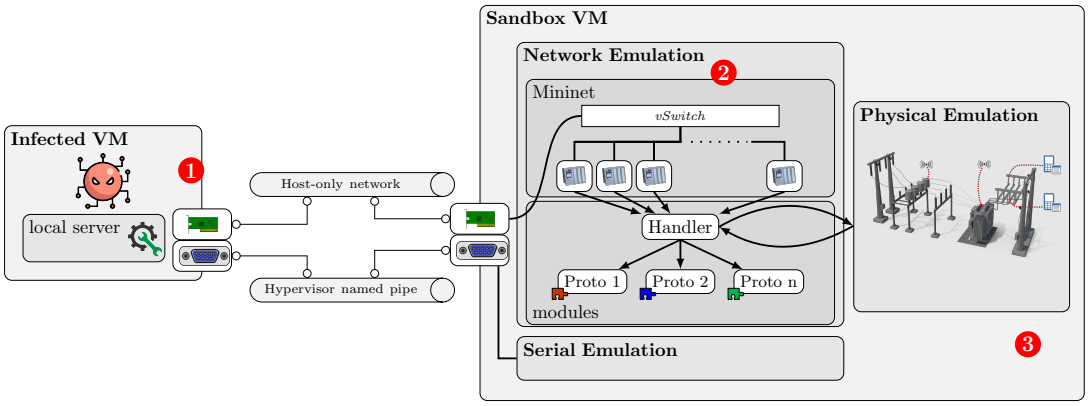


Figure 7.2: Sandbox architecture.

Since one of our requirements is to support various industrial control protocols, albeit not necessarily at the same time, each simulated host must be able to dynamically enable or disable the support for an industrial control protocol. Moreover, the network topology and simulated hosts must be configurable. Therefore, there needs to be a component that parses the configuration requirements and

uses each supported protocol as required. Since this component will essentially handle the information flow of the simulated industrial device, we could tie this component to the physical simulation, requesting and updating values from the simulated physical process and translating these values into the industrial control protocol.

With this information handler in mind, we can implement the different supported industrial control protocols into modules that the handler can dynamically import to enable a specific protocol and communicate with the infected machine or any standard industrial control client.

This handler presented the primary implementation challenge. Once the sandbox instantiates a device, the virtual device must follow with an instantiation of this handler. Therefore, each virtual device implementation must have a unique handler that bridges the device with the different network protocols and the physical simulation. Following our flexibility and extensibility requirements, these handlers must be imported and instantiated at runtime. To overcome this complexity, we implemented a generic launcher that checks the device and handler modules against the provided configuration at runtime and creates the required instances. Using this launcher component, we do not restrict the implementation to a particular scenario and allow the sandbox to be extended to include additional physical processes and devices.

So far, we have implemented modules for handling IEC-60870-101, IEC-60870-104, and Modbus TCP using Scapy, IEC-61850 using the iec61850bean library, a Matrikon OPC server for OPC-DA, and CPPPO for Ethernet/IP CIP.

IEC-60870-101 (Serial) and IEC-60870-104 (TCP/IP) are remote control protocols operators use to control multiple substations from a central control room. Therefore, this scenario simulates the remote communication between an infected

host in a control center and multiple remote substations linked by a WAN or a legacy serial dial-up connection. We use the IEC-61850 module to simulate devices within the local LAN of a substation. Thus, the infected machine in this scenario would be a local operator workstation inside the substation within the local LAN.

For the physical process simulation, we can either have one virtual host do all the simulations and coordinate with the other hosts or have each host simulate part of the physical process and synchronize the simulation state with the relevant hosts. In either case, we need the simulated hosts to communicate with each other and synchronize the state of any physical sensors and actuators handled by the simulated devices. This physical communication must be: (1) **Fast**, meaning that the physical process simulation must be as realistic as possible and run in real time. (2) **Up-to-date**, meaning that each device holds only the most recent data. (3) **Concise**, meaning that we only send the necessary values, and (4) **Addressable** as we want the values to reach the desired device.

A UDP protocol is sufficient for the **fast** and **up-to-date** requirements, as there would be no issue with a missing packet if a device receives a more recent packet, and the lack of flow control offers a speed advantage over a TCP protocol. As for **conciseness** and **addressability**, a fixed-size structure of bytes carrying the necessary information fulfills this requirement. We designed a simple small payload that allows us to transmit concise sensor and actuator values to and from the simulated devices, with addressable device IDs, as most industrial control protocols do.

Our proposed solution is a UDP protocol that uses fixed 28-byte packets containing sender and receiver IDs, a message type, two integer values, and two float values. Sender and Receiver IDs make the protocol addressable in the applica-

tion layer, meaning that the sender can broadcast the packet without knowing the recipient's IP address and effectively transmit a value, which is helpful in scenarios in which we dynamically build a network topology without knowing the IP addresses that Mininet assigns to the hosts. The message type allows us to customize the messages and specify the packet's data, be it a range, set-point, sensor value, or actuator value. Since we want a standardized way of sending the data, we always send a fixed-size payload, and it is up to the message type to determine the semantics of the four values.

Using this protocol, we construct the physical simulation in separate modules that we instantiate as needed. These modules use the protocol to communicate with the simulation handlers, providing them with the up-to-date state of the sensors and receiving any state changes for the actuators to simulate the modifications in the simulation.

7.2 Deployment tests

A command injection attack happens when an unauthorized party sends a command to a Remote Terminal Unit (RTU). To execute this command injection attack, we take some ideas from the behavior of Industroyer 1 to create malicious software that runs on a compromised device in the SCADA network. The malware launches attacks in stages from this compromised device: The first stage corresponds to a reconnaissance state in which it scans the network to identify any live hosts and then determines which are RTUs. Following this initial reconnaissance stage, the malware connects to the identified RTUs, posing as a legitimate SCADA server, which allows the malware to receive the polled measurements and status from all the identified RTUs. The malware extracts and accounts for all the IOAs in each RTU by receiving the reported status data. After gathering enough

information from the RTUs to map out all the circuit breakers in the system to each RTU, it executes the final stage of the attack by sending commands to each RTU with a circuit breaker that instructs the RTU to open the breakers, resulting in a massive blackout.

We use a Python script to simulate our malware. The script runs in an additional node in mininet representing the compromised device (e.g., a compromised operator's workstation). Once the execution starts, the script identifies live hosts by sending ARP probes destined for each host in the compromised machine's subnet. Once identified, the malware probes each live host with a stealth SYN scan (A half-open TCP connection that closes the socket before the three-way handshake is complete) for the specific TCP port 2404 designated for IEC104 communications. If a live host is an RTU, the TCP socket will receive a TCP SYN-ACK packet; otherwise, it will receive a TCP RST packet. With this simple technique, the malware can ascertain whether a particular endpoint will likely be an RTU.

After this, the script opens a socket with each RTU and sends a 'STARTDT act' packet, allowing the malware to receive the IEC104 polled measurements and states, looking for any RTUs that report a type-3 ASDU, which corresponds to a circuit breaker in the context of a power grid. For each type-3 ASDU, the malware uses the same scapy-based IEC 104 module to extract the IOA corresponding to the particular circuit breaker the RTU is reporting, mapping out which RTUs have circuit breakers that could be compromised. After the malware finds no new circuit breakers in the network, it sends a type 45 ASDU (Single Command) for each identified breaker, instructing each corresponding RTU to open it. By doing so, and because our simulated power grid is a radial circuit, the load node is "physically disconnected," causing a simulated blackout.

The authors in [30] also propose a series of compelling deception attacks in an emulated power grid system. Although the assumed adversary shares the ARP poisoning technique to obtain network access in their and our attack scenarios, our attack design differs from theirs in terms of the industrial protocol, the power grid component, the threat model, and the consequences of the attack.

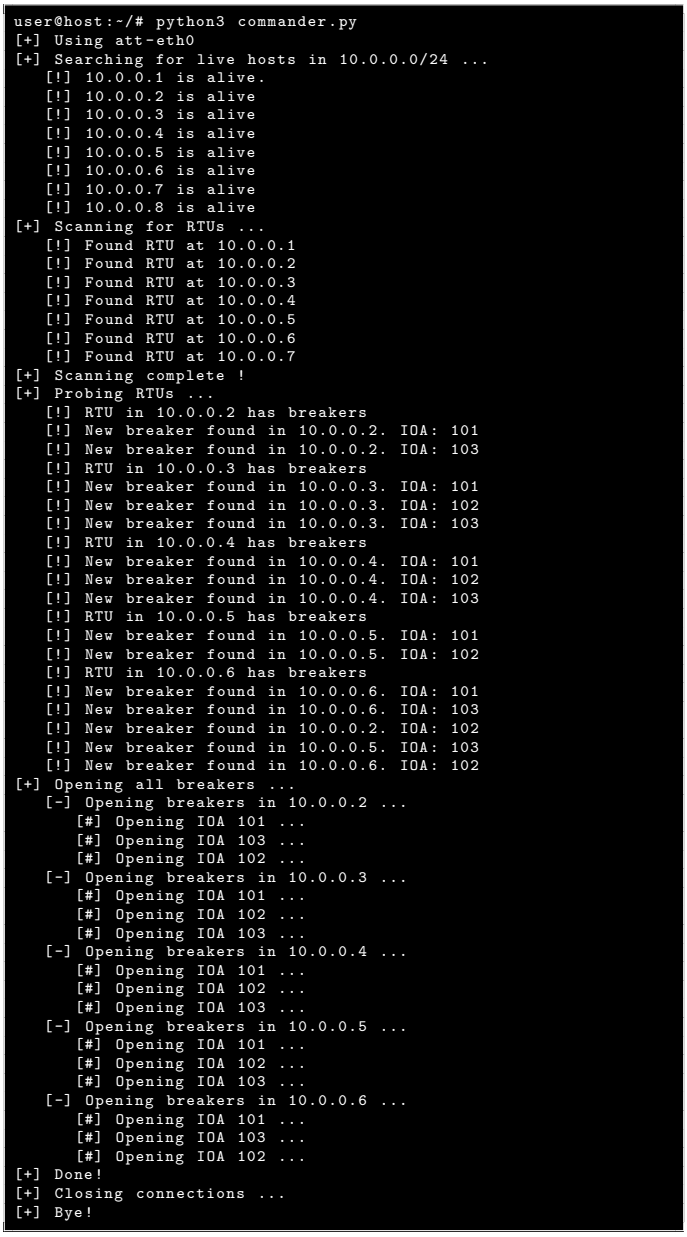


Figure 7.3: Execution of the simulated malware performing a command injection attack.

Chapter 8

Studying a malware with the framework

Table 8.1: Malware samples.

Sample	Industroyer	SHA-256
101.dll	1	a319551ef72492b3cd489de676b2f6e7938a5ef23e572d36dd742b599686caac
104.dll	1	7907dd95c1d36cf3dc842a1bd804f0db511a0f68f4b3d382c23a3c974a383cad
61850.dll	1	4e7d2b269088c1575a31668d86de95fd3dde6caa88051d7ec110f7f150058789
haslo.exe	1	ad23c7930dae02de1ea3c6836091b5fb3c62a89bf2bcfb83b4b39ede15904910
opc.exe	1	156bd34d713d0c8419a5da040b3c2dd48c4c6b00d8a47698e412db16b1ffac0f
svchost.exe	1	7cc9ac6383437dd96161b93b017500a22a2c8d05f58778b9b9fce8ea73304546
40_115.exe	2	d69665f56ddef7ad4e71971f06432e59f1510a7194386e5f0e8926aea7b88e00

We summarize our Industroyer samples in Table 8.1, providing the SHA-256 signatures of each payload. The curious reader can verify the signatures via third-party services such as "virustotal" to corroborate the authenticity of the samples. Industroyer 1 consists of a set of DLLs and executable files. In contrast, Industroyer 2 is a single executable file. Both malware versions are post-exploitation tools (final payload) deployed after a successful intrusion.

8.1 Methodology

To understand the malware capabilities, we studied each payload sample through three sequential stages: static analysis, dynamic analysis, and semantic analysis.

Static analysis: Initial sample exploration to gather sufficient information about each malicious image without executing them. Some of the tests performed involve strings extraction, gathering of import/export functions, and detection of anti-debugging mechanisms (e.g., obfuscation/encryption). In this stage, we determined various execution constraints, such as the file format of the expected configuration files for specific payloads, a control flow graph analysis to determine the predicted values by the malware, and entry points for further debugging.

Dynamic analysis: Behavioral study of the sample. The objective is to understand how the malware behaves when executed. Based on the execution constraints obtained from the static analysis, we deployed an isolated environment that simulated the scenario expected by the malicious software. The output of this stage provides information about the interaction between the malware and its environment (e.g., packet captures, log files, etc.). It helps us to craft adequate packets for the modules in scenarios where the malware waits for certain values and packet sequences.

Semantic analysis: Semantic study of the sample. The objective is to understand the real impact of the malware execution against the physical system. We simulate physical systems and electricity network services in our testing environment using the modules we designed for the framework to determine how the malware can affect the targeted infrastructure.

8.2 Attack Patterns of Industroyer 1 & 2

Being a modular industrial control malware, Industroyer 1 can carry out multi-pronged and multi-target attacks against different processes and parts of the power grid operation by using four payload modules, each targeting a specific power grid control protocol: IEC 101, IEC 104, IEC 61850, and OPC Data Access (OPC-DA). In contrast, Industroyer 2 is a stand-alone executable that uses a single protocol to perform its attack: IEC 104. Each supported protocol also reveals the potential industrial control targets the attackers could compromise. Using Industroyer 1, an adversary could launch an attack against an OPC-DA server, RTUs via IEC-101 or IEC-104, and substation IEDs using IEC-61850. As for Industroyer 2, the targets are RTUs supporting IEC-104.

Given the flexibility of this malware, it can compromise an operator's workstation pretty much anywhere within the confines of the power grid network and still be effective against the system. Figure 8.1 shows the attack surface of the malware and the different interactions it can employ to disrupt the system. It has a module to interact with substations via the old serial standard IEC 101, another module to use the modern TCP-based IEC 104, a module to interface with substation devices using IEC 61850 directly, and a final one to interact with legacy OPC servers.

Figure 8.1 shows how Industroyer 1 could have interacted with several devices in the power grid; e.g., it could be launched from the SCADA computer in the central control room and use serial connections (such as IEC 101) or TCP/IP connections (such as IEC 104) to reach substations. Alternatively, it can be deployed in a workstation within a substation and use IEC 61850 to discover and target IEDs. As far as we know, this is the most detailed network architecture representing the capabilities of Industroyer 1 and 2 and the specific locations

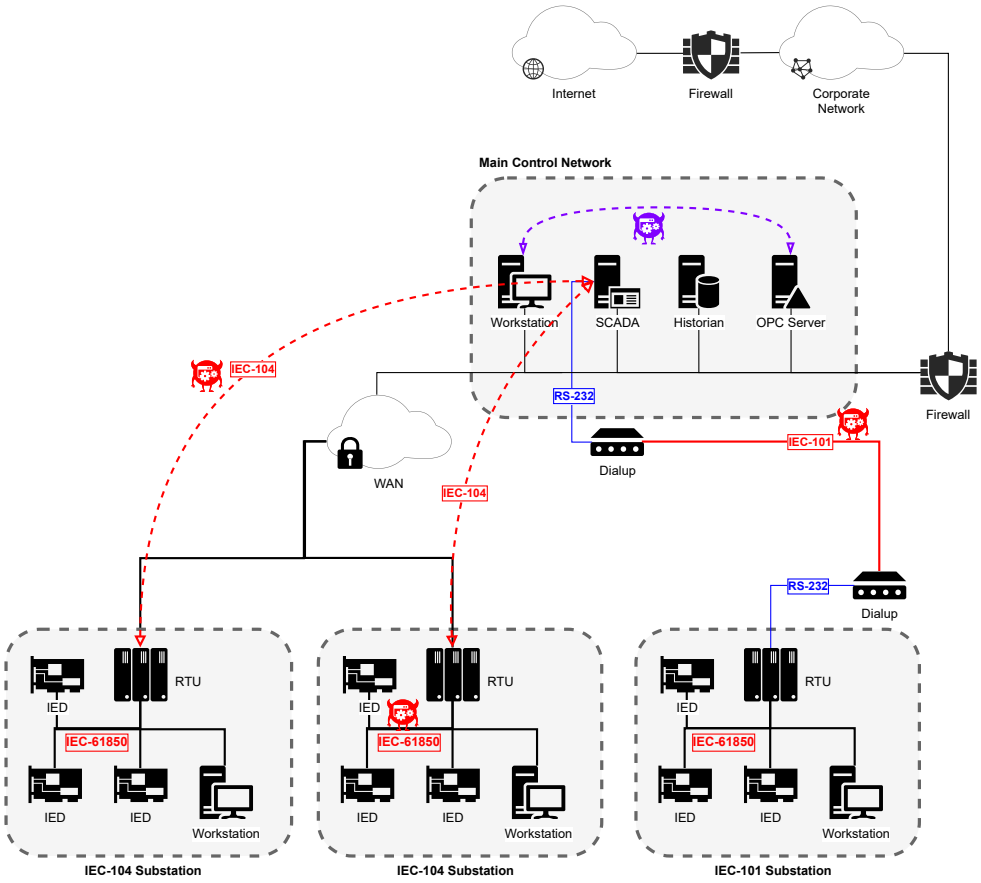
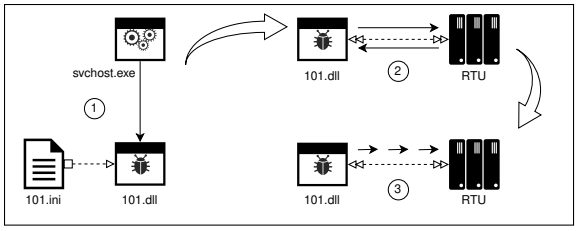


Figure 8.1: Industroyer's attack surface

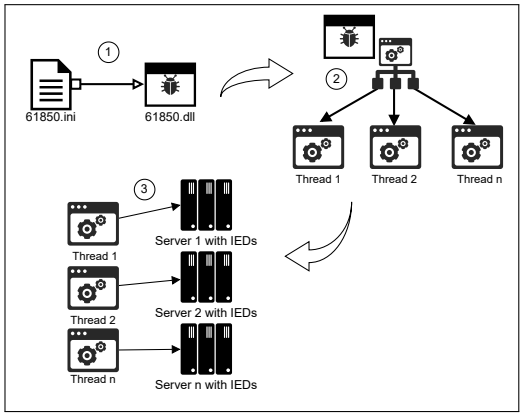
where they could interact with power grid equipment. This diagram also helps us design the scenarios we will use in the sandbox to fool Industroyer 1 and 2 into believing they are inside a power grid.

Figure 8.2 shows the overall process each payload executes to accomplish its task. While there are specifics to each payload, we found that the overall process follows the same pattern.

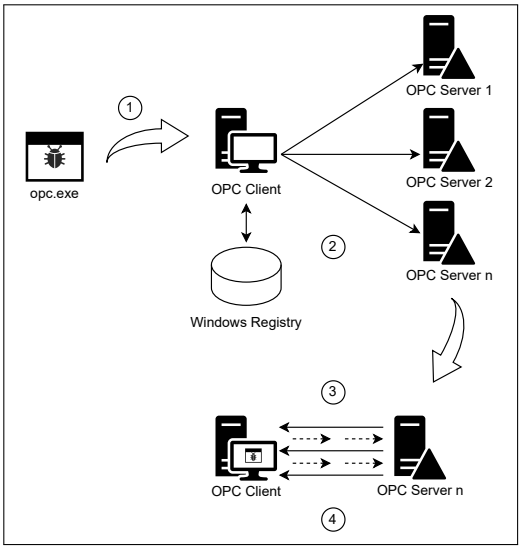
Targets. This is the step where the payloads differed the most. In some cases, the attacker had a lot of information about the specific targets and baked the targets into the executable (e.g., Industroyer 2); in other cases, the malware expected a specific target (e.g., IP addresses, ASDU Common Address, and specific IOAs)



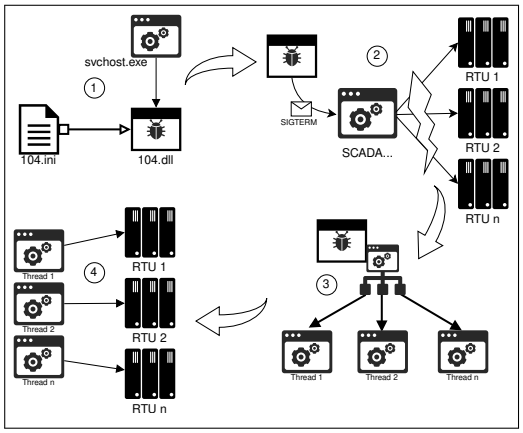
(a) IEC-101 payload



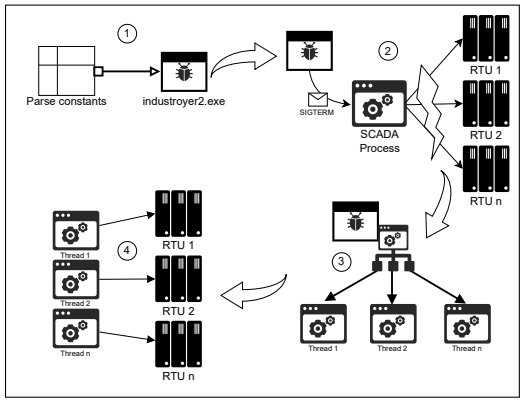
(b) IEC-61850 payload



(c) OPC-DA payload



(d) IEC-104 payload



(e) Industroyer 2 payload

Figure 8.2: Overall malicious process for each payload

but this information was provided via a configuration file read at execution time (e.g., Industroyer 1's IEC 104 payload), finally, in some cases the malware did not have nor expected any information about the target; instead, the malware executed commands to perform a reconnaissance task to identify potential targets (e.g., Industroyer 1's OPC-DA payload).

Blocking. Most payloads prevented a legitimate process from interfering. If the legitimate control process retains a communications channel with the target, it can prevent or override any malicious command. Each payload attempts to avoid this by terminating the legitimate process (e.g., the IEC-104 payload terminates the running SCADA software communication agent) or locking the required resource within the operating system (e.g., the IEC-101 payload blocks all serial ports).

Connection. Most payloads use a TCP-based protocol to send malicious commands to the targets. These payloads create a separate thread to handle each TCP connection (e.g., IEC-104) or occupy the serial port (IEC-101).

Exploit. In this step, the malware sends malicious commands to devices in the power grid. Some payloads send specific commands to each target (e.g., the IEC-61850 payload sends 35 commands to a CSWI). In contrast, others enter an infinite loop in which they periodically send malicious commands (e.g., the IEC-104 payload of Industroyer 1 keeps repeating a configurable sequence of commands forever).

Regardless of the specific payload nuances, the main objective of all payloads is to change the status of particular circuit breakers controlled by RTUs or IEDs.

8.3 Malware payloads

To study each payload, we configured the framework according to the necessary network protocols and modules for the payload to execute correctly. By executing the payloads in a virtualized, isolated, and controlled environment, we gained particular insights for each payload. We now summarize our findings for each one of the payloads.

8.3.1 IEC 101 payload

The IEC 60870-5-101 payload communicates with the target RTU via a serial port in the compromised host. As shown in Fig. 8.2a, the overall process has three primary stages: ① Initialization and configuration, ② Establishing the communication and resetting the serial link, and ③ malicious control loop. It uses the FT1.2 frame format over the serial port to communicate with the target. Figure 8.3 shows the structure of such frames.

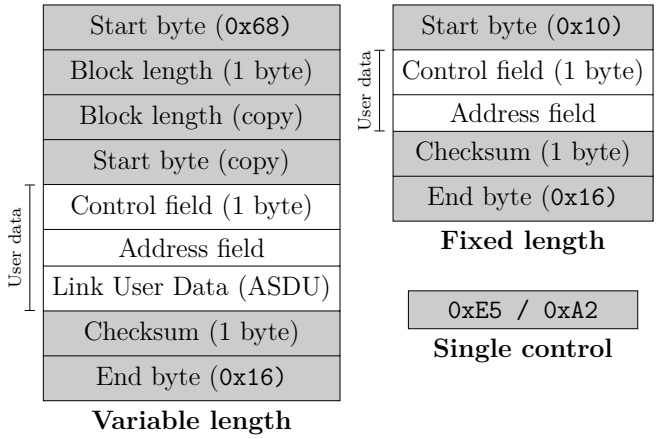


Figure 8.3: Description of standard frame format FT1.2

This payload requires a configuration file containing the serial port to use, the remaining ports to occupy, the name of the legitimate process, the RTUs' Common

Address, and the target IOA ranges. In the first execution stage, the payload reads the configuration file, terminates the legitimate process, and occupies the configured serial ports.

The second stage involves establishing a proper connection with the target RTU. The payload sends a "Reset link" command encoded in FT 1.2 format to the RTU through the serial port, re-establishing a clean communications channel with the target and resetting the send-receive counters related to the IEC-101 protocol.

In the final stage, the payload enters an infinite loop, periodically alternating between sending a single (C_SC_NA_1) and a double (C_DC_NA_1) command with the 'OFF' value. This payload follows the Select-Before-Operate paradigm by sending two frames per command, one for selecting the information object and the other to execute the command.

It then continuously sends an 'OFF' value in an attempt to open the circuit breakers in the target substation. The payload will send a single and a double command for each circuit breaker in the configured IO range, infinitely iterating this range. An educated guess as to why attackers alternate between single and double commands is that they were unsure which type of command the various IOs in the target substations support; thus, they send both.

8.3.2 IEC 104 payload

This payload requires a configuration file containing several target parameters: IP addresses, ASDU Common Addresses, IO Address ranges, TCP port, and time-outs. In addition to these parameters, the attacker can fine-tune the next stage with additional parameters that control the qualifier of the IEC-104 command, whether the malware will alternate the value of each command, and the initial

value to use.

In this first execution stage, the payload reads the configuration file and identifies all the target RTUs with their corresponding attack parameters. After this, a second stage involves ending the legitimate connections by terminating the legitimate process, if any. In the third stage, the malware creates a thread for every target RTU, prepared with all the necessary parameters to establish a connection with its target and the corresponding attack. Given the configuration, each thread will vary the actions performed during the malicious loop according to the attack parameters.

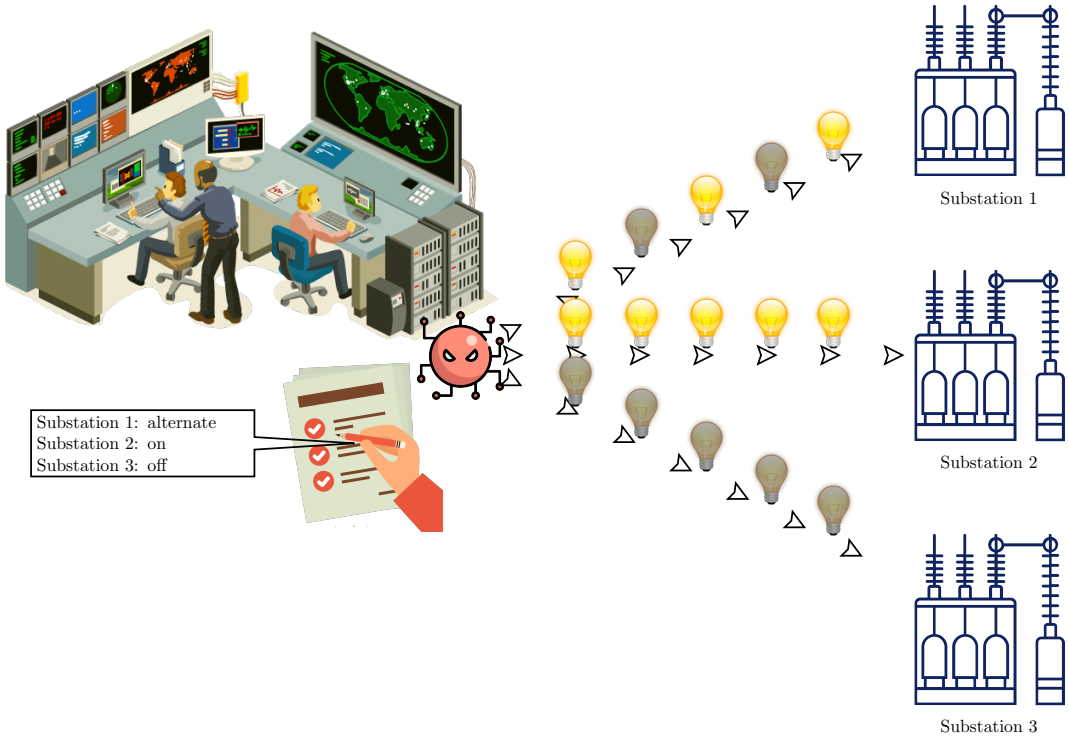


Figure 8.4: Differing attacks by IEC 104 in Industroyer 1.

Following the attack configuration, the payload enters an infinite loop in the final stage. This payload loosely follows the IEC-104 standard in that it only

sends the minimum necessary messages to communicate with the targets without strictly following an expected procedure of a legitimate connection. Moreover, this payload disregards any messages the target sends, continuing the attack even if the target RTU replies with an error frame. The attack itself varies according to how the adversary configured it. It can send the same value (ON or OFF) in every IEC-104 frame or alternate between values. This is the only payload that allows that level of configuration. All other payloads have a fixed set of instructions for the circuit breakers: (send repeated open commands, such as IEC 101, or alternate between open and close 36 times as IEC 61850). With the IEC 104 payload, the attacker can select to send continuous open commands (to keep overriding a manual close command), send repeated close commands, or send an alternating set of commands (open, followed by close) in an infinite loop.

Overall, the IEC-104 payload had the most advanced configuration file. As shown in Fig. 8.4, the attacker could create different configurations that allowed the malware to interact differently with each device and each substation. For example, it could ask one device in a substation to alternate its status and another in the same (or a different substation) to remain open.

8.3.3 OPC

The OPC payload of Industroyer 1 aims to find manipulable devices using known standard COM interfaces and close and immediately open enumerated circuit breakers to cut off power. Our analysis shows that the malware attacked the OPC client station by initially retrieving the registered OPC servers in its Windows registry. It obtains all items in these servers and finally looks for items with a specific string identifying the targeted circuit breaker. Once it finds those items, it uses the exact commands to select and execute a disconnection from the

power grid. As shown in Fig. 8.2c, the execution of the OPC payload is as follows:

① *Initialization and configuration:* The OPC module's entry points call a single primary function that handles the entire logic of the executable. Without requiring a configuration file, as some of the other payloads, the OPC payload's primary function starts by enumerating all OPC servers in the Windows registry by looking for the OPC Server 2.0 CATID, a globally unique identifier (GUID) that distinguishes between types of interfaces.

② *Establishing communication:* The payload uses the Component Object Model (COM) interface `IOPCBrowseServerAddressSpace` to find locally registered OPC servers. An OPC group needs to be created for each server to interact with and manipulate items. The function invokes the `IOPCServer::AddGroup` method to achieve this. Then, to enumerate the OPC items in each server, Industroyer 1 leverages the `IOPCItemMgt` interface.

③ *Retrieving OPC items:* The payload looks explicitly for items containing one of the following strings: `ctlSelOn`, `ctlOperOn`, `ctlSelOff`, `ctlOperOff`, and `stVal`. According to ESET [8], these are strings associated with OPC items for *ABB* devices, which are used as abstractions of the circuit breakers. During the server enumeration, the primary function logs the name of each server. Assuming all the strings are present, the function will begin by logging the name, quality, and value of `\Pos.stVal`. The OPC item `\Pos.stVal` holds the current status of the circuit breaker, which can have one of four integer values: **0 for intermediate-state, 1 for off, 2 for on, and 3 for bad-state**. The rest of the items are functions that execute specific commands in the physical breaker.

④ *Writing OPC items:* The payload uses the `IOPCSyncIO` interface to write `0x01` bytes to `ctlSelOn` and `ctlperOn` to close the circuit breaker the given item refers to. The module then logs the new values and uses the same interface to

write `0x01` bytes to `ctlSelOff` and `ctlOperOff`, opening the circuit breakers. The payload retrieves and logs the final status after the last write.

In typical scenarios, to close a circuit breaker (thereby activating it), one must first select the circuit breaker by sending a `0x01` byte to the `ctlSelOn` item. Then execute the task by similarly sending a `0x01` byte to the `ctlOperOn` item. To open a circuit breaker (which would be the objective of a malicious actor trying to cut off power), one would have to send a `0x01` byte to `ctlSelOff` and `ctlOperOff`. Industroyer 1 performs these actions automatically, where the values of `ctlOperOn`, `ctlOperOff`, `ctlSelOn` and `ctlSelOff` are overwritten to 1.

The module writes a log file during its operation and prints the status of the *stVal* item, which, as we previously mentioned, could represent the resulting circuit breaker status. The payload logs the values of the *stVal* item in different stages of its execution: The initial value is logged under *State: Before*; The second value after setting `ctlSelOn` and `ctlOperOn` under *State: After ON*; and the third value after setting `ctlSelOff` and `ctlOperOff` under *State: After OFF*. According to our tests, the logging and value writing will still work in case some items are missing; however, we believe that the malware requires all of these items to be present to impact the physical circuit breakers.

8.3.4 IEC-61850 Payload

Industroyer 1 targets MMS, a protocol used to connect with IED devices. Unlike other protocols, IEC-61850 is deployed within the substations themselves. Thus, understanding its functionality requires additional insight into the inner workings of a substation.

Fig. 8.5 illustrates a general IEC 61850 network in a substation. The 61850 payload has three primary stages, as shown in Fig. 8.2b:

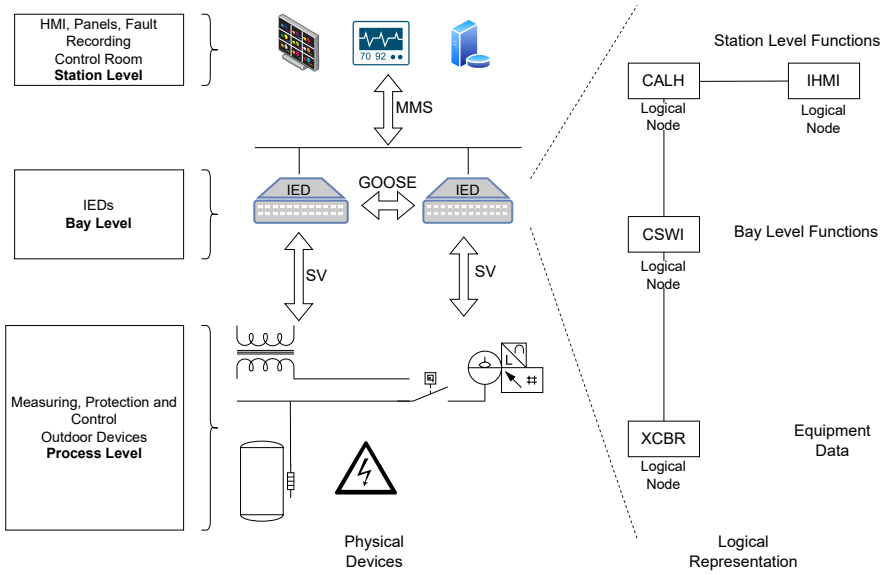


Figure 8.5: A power grid substation network.

① *Initialization and configuration:* The payload's entry point may have a configuration (.ini) file with a list of target IP addresses. In this stage, the payload reads the configuration file and identifies targets with open port 102 (MMS service); otherwise, it scans the local network for viable targets.

② *Establishing communication:* In this stage, the payload creates an individual thread for each IP address with an open 102 port to establish the connection and launch the attack. It sends a TCP connection request and a Connection-Oriented Transport Protocol (COTP) request. If the target accepts the COTP request, the malware sends an MMS connection initiation request. The target confirms the connection initiation by sending an MMS initiation response and establishing a successful connection.

③ *Malicious control loop:* In the final stage, the payload enters into a loop of write requests. The goal of the 61850 payload is to locate the control switches (CSWIs) exposed by the target. The malware targets these switches and changes the values of their control variables.

Devices (IEDs) that support IEC 61850 are typically factory-configured with *data models* from the manufacturer. These data models summarize the device's capabilities. For instance, if an IED can protect a circuit from over-current, it will have a logical node named `PTOC` (a logical node beginning with a "P" means it is for protection). Similarly, a device with over-voltage protection will have a logical device named `PTOV`.

Industroyer 1 uses the command `getNameList` to find the logical devices of a given IP address supporting IEC 61850. In particular, Industroyer 1 requests if the device has logical nodes named `CSWI`. This logical node is used for control (logical names starting with "C" means that they are control devices). These control elements can operate circuit breakers (`XCBR`), isolators (`XSWI`), or other process equipment located in a merging unit in the switchyard (the process level in Fig. 8.5). Logical node names starting with an "X" means that these devices are switchgear and are located in the switchyard.

In the analysis of the 61850 payload, we observed that Industroyer 1 tries to identify the following CSWI objects:

stVal: This variable indicates the status of the CSWI. It can assume several possible values, among which 0x40 is for position ***OPEN*** and 0x80 ***CLOSED***. In our analysis, we found that any value that is not 0x80 is (incorrectly) considered as ***OPEN*** by Industroyer 1.

ctlModel: Defines the mode in which the CSWI operates. In mode 0, it only reports its status; in mode 1, it can be controlled directly by a single command; in mode 2, it is configured with **Select Before Operate (SBO)**, where (for safety reasons) the switch first needs a *select* command indicating the intention to change its status, and only after it receives a second command (operate) it changes its value. In mode 3, it can be controlled by a single command, and

the IED confirms that the physical device changed its position as requested (or not). Finally, in mode 4, the switch is in SBO mode, and in addition, it sends the confirmation command. We configured the IED server to reply with all of these possible values but found that Industroyer 1 considered `ctlModel` replies of 0, 1, 2, or 3 as if the server was configured with a `ctlModel` of 1 (i.e., Industroyer 1 sends an Operate command without a Select beforehand).

After the initial reconnaissance is completed, Industroyer 1 delivers its malicious commands by sending ***write requests*** based on the `ctlModel` value. If `ctlModel=4`, then it sends two write requests: (1) **SBO** to select the device before sending Operate command, and (2) **Oper** to change the control value of CSWI. For any other value of `ctlModel`, it sends only one write command, **Oper** (which, as we mentioned before, violates the standard). We infer that the machine targeted by Industroyer 1 was only either in a `ctlModel` mode of 1 or 4, and that is the reason the malware developers did not create an accurate payload for the reply if the server replies with a `ctlModel` of 0, 2, or 3.

Another interesting observation is that the malware sends a sequence of 36 commands, each time alternating the status of the circuit breaker. This repeated opening and closing of the circuit breaker may be meant to damage the circuit breaker or associated device (in addition to trying to create a blackout).

8.3.5 Industroyer 2

Unlike its predecessor, Industroyer 2 does not use a custom configuration file through the command line to determine the targets and nature of the attacks. Instead, the malware has hard-coded configurations that specify the different targets. Having the targets hard-coded means the attacker must re-compile the malware whenever they need to change the attack targets. Moreover, since the

adversary needs to have this information at compilation time, we assume that the attackers had prior detailed knowledge about the target infrastructure (IP addresses of target RTUs and specific IOA addresses of the circuit breakers).

We summarize the overall process in the same four basic steps, shown in Fig. 8.2e: ① the malware parses the configuration from the hard-coded strings, ② it kills the legitimate process and any active connections with the targets, ③ it launches an individual thread for each target RTU, and ④ it executes the malicious process over a newly established connection with each target RTU.

Upon execution, the malware first checks the command line arguments to determine the value of two possible arguments: `"-t"` and `"-o"`.

Including the `"-t"` flag will set a timed start of the process. If the attacker includes this argument as part of the command line, the malware will take the following token in the command line and use it to determine when the process begins. The minute specified by the parameter determines this time. For instance, if the adversary runs the malware using the value "`-t 40`", the malware will wait until the next 40th minute in an hour. If the attacker runs the command, say at 02:35, the process will begin at 02:40; if s/he runs it at a minute greater than the specified, it will wait until the next occurrence of the specified minute within the following hour. For instance, if s/he runs it at 02:55, the process will begin at 03:40.

As for the second possible argument, the meaning is relatively straightforward; the `"-o"` parameter determines the filename of a log file the malware will use to store the output. If this argument is not specified, the malware prints any output on the standard output.

As a result of the static analysis we performed on the sample, we determined that an array of strings comprises this hard-coded configuration, each string con-

taining the necessary values to execute an attack against a targeted RTU. As an example, the configuration string located at offset `0x9818` is `u"192.168.122.2 2404 2 0 1 1 PService_PPD.exe 1 \"D:\\OIK\\DevCounter\" 0 1 0 0 1 0 0 8 1104 0 0 0 1 1 1105 0 0 0 1 2 1106 0 0 0 1 3 1107 0 0 0 1 4 1108 0 0 0 1 5 1101 0 0 0 1 6 1102 0 0 0 1 7 1103 0 0 0 1 8 "`.

In the sample we analyzed, the malware targets three IP addresses, each with a specific list of targeted information objects (target circuit breakers). The target systems exist within at least two subnetworks, one of which could be a subnetwork with a 23-bit prefix (Fig. 8.6).

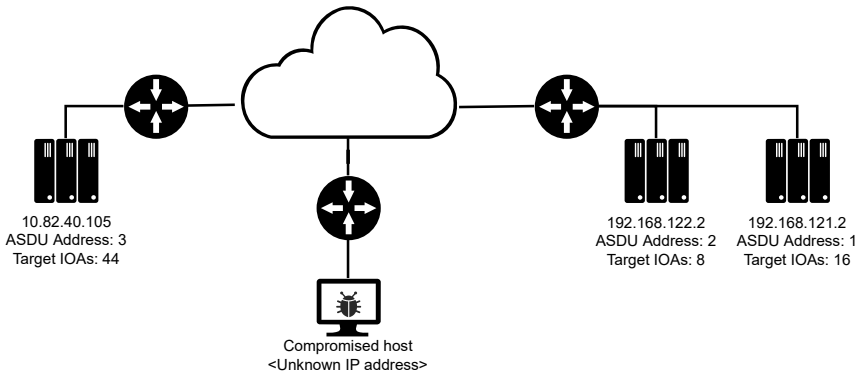


Figure 8.6: Targets of Industroyer 2.

Table 8.2 shows the specific targets. We infer the attacker targeted three RTUs (presumably three different substations). In the first substation, it attempted to open 16 circuit breakers; in the second substation, it attempted to open 8 circuit breakers; and in the third substation (the largest substation with 44 circuit breakers), it attempted to open 28 circuit breakers, and then it attempted to change the status of 16 circuit breakers from open and then to close. Opening this large amount of circuit breakers in three different transmission substations can significantly negatively impact the power grid.

Notice that the attacks included always disconnecting the circuit breaker, al-

Table 8.2: Industroyer 2 configured targets

Target IP	ASDU Address	Default command Additional inverted	Target IOAs	Single Command	Double Command	Select Before Operate	Invert default
192.168.121.2	1	0 0	16	0	16	0	0
192.168.122.2	2	0 0	8	0	8	0	0
10.82.40.105	3	0 0	44	28	16	0	28

ways connecting the circuit breaker, and connecting and disconnecting repeatedly. This last attack appears similar to the Aurora generator attack [73], where connect and disconnects can be used to apply maximum torque.

8.4 Malware evolution

Having access to both variants of the malware, used years apart, allowed us to compare both samples and gain some insights into the evolution of the attack. Industroyer 1 does not follow the standard in many ways. Not only did it disregard any incoming packets, but it also did not behave as a legitimate client. Industroyer 2 fixed these issues by following the correct behavior of the standard. While Industroyer 1 can be detected by monitoring the expected protocol behavior, Industroyer 2 would bypass this anomaly detection check.

Fig. 8.7 shows the expected behavior of a controlling station in IEC 104. Under normal circumstances, the controlling station would establish a TCP connection with the RTU, poll for the existing devices with an interrogation command, and send keep-alive frames to maintain the connection with the controlled station

while receiving measurements.

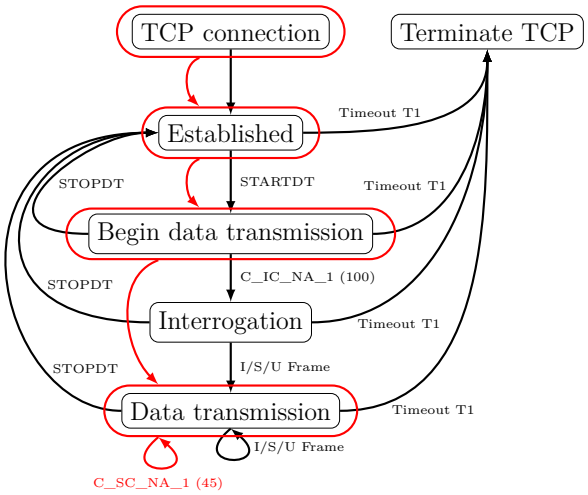


Figure 8.7: IEC-104 connection behavior. Industroyer 1 (red) does not follow the expected behavior of a legitimate connection (black).

The behavior of the IEC-104 payload of Industroyer 1 varies from the behavior of its successor. Industroyer 1 skips the interrogation stage, disregards any incoming frames and begins to send commands after starting the data transmission at a rate of 2 frames/s in an infinite loop for as long as the process executes. Industroyer 2, however, follows the standard more closely; not only does it verify every received frame as a valid IEC-104 frame, but it follows the legitimate control flow of sending an interrogation command prior to any other interaction with the controlled station after the data transmission begins, sending commands at a rate of 0.2 frames/s. Also, Industroyer 2 does not execute indefinitely like the 104 and 101 payloads (or 36 times in the case of the 61850 payload); it sends one (or two) commands to the configured targets and gracefully terminates the connection with the controlled station after sending the malicious commands.

Looking at the evolution of these attacks, we see the attackers constantly changing tactics in attempts to trip devices. First, Industroyer 1 was a Swiss army knife to attack the power grid. It was developed without any specific target,

and it could deal with legacy connections (IEC 101), modern connections (IEC 104), and interoperability systems (OPC). In addition to working on SCADA computers in a central control room, Industroyer 1 could even be deployed in the computers at substations (IEC 61850).

In addition, Industroyer 1 was highly configurable. It expected configuration files telling it how and what to attack. It also attempted a limited reconnaissance of potential victim devices. Finally, it had several bugs; most notably, it did not strictly follow the industrial protocol specifications, and its success may have depended on how tolerant victim devices were to non-standard industrial protocols.

Because Industroyer 1 was so adaptive, without the configuration files, we do not know the targets or the attacks launched (did they open and close the circuit breakers? Only some of them? Open and then close? etc.) In contrast, Industroyer 2 was a targeted strike; it did not read any configuration files, so attackers knew how and what to attack before deploying the malware. It also only targeted one industrial protocol (instead of four.) Industroyer 2 was also fairly polished, following the expected behavior defined by the industrial protocol standards more precisely. These changes might mean that the attackers did not want to share all their capabilities with the forensic teams and that the malware was now polished enough to interact with most devices without errors. As time passes, attack tools may become more polished, and the teams using them will only use the pieces necessary for the attacks.

Chapter 9

Extending the framework

A honeypot is an imitation information system designed to mimic genuine network services. Its purpose is to trick potential attackers into engaging with it, capturing and recording each interaction. This collected data helps researchers better understand and analyze attacker behavior.

To gather information about novel threats to ICS, honeypots can be used to identify new attack patterns to help us prepare better protections. The first ICS honeypots, such as the SCADA HoneyNet Project [47], were initial efforts to provide low-interaction simulations limited to one or two network protocols commonly used in Industrial Control. Newer ICS honeypots such as HoneyPLC [38] extended previous work to offer a high-interaction environment and support for more ICS protocols and devices.

Developing general-purpose ICS honeynets presents numerous challenges due to the diverse range of vendors, industrial protocols, physical processes, control devices, and functionalities involved. Previous efforts have grappled with accurately replicating the varied nature of these systems while maintaining a realistic environment. By expanding our framework, we have introduced a cutting-edge hybrid-interaction honeynet that represents a significant advancement in the field

of ICS honeynets. Leveraging our framework's modular architecture, we seamlessly integrate additional high-fidelity physical process simulations, industrial control protocols, and precise device fingerprints through a profile engine.

9.1 Interaction levels

Low-Interaction. Low-interaction honeypots are a type of cybersecurity resource that provides adversaries with minimal functionalities. Typically, they are implemented using uncomplicated scripts. These honeypots offer two distinct advantages: they are easier to develop and maintain, and their limited functionalities pose a lower risk of adversaries gaining control over them. However, the primary drawback is that adversaries may abandon their attack before realizing they are interacting with a deceptive resource.

High-Interaction High-interaction honeypots are designed to simulate interactions that closely mimic those of a natural system [3]. They can be implemented using genuine devices or through software emulation, such as virtualization. In the context of ICS, high-interaction honeypots excel at simulating attacks in a highly realistic setting compared to low-interaction honeypots. This is made possible by their ability to provide a higher fidelity in simulating physical processes, thereby creating a more authentic environment for analyzing and responding to potential threats.

High-interaction honeypots, while offering advanced interaction capabilities, come with notable drawbacks. Firstly, their deployment and maintenance expenses are quite high. Secondly, there is a substantial risk of an adversary taking control. The considerable freedom provided by high-interaction honeypots could be exploited by an adversary to commandeer the honeypot, lock out the administrator, and utilize the honeypot to establish a foothold in the network [68].

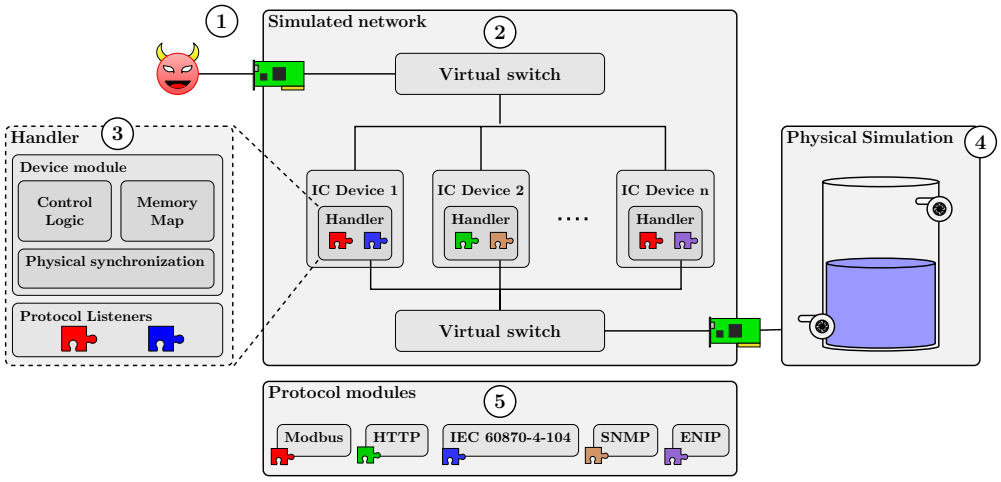
Despite their superior interaction capabilities, caution is advised when using high-interaction honeypots due to the significant risk of adversary takeover.

Hybrid-Interaction Hybrid-interaction honeynets combine the advantages of both low and high-interaction honeypots, as referenced in [13, 17]. A key advantage of these hybrid-interaction honeynets is their flexibility. They enable the integration of multiple types of honeypots within the same honeynet, allowing for a mix-and-match approach to meet the specific honeynet requirements. Moreover, the administrator of a hybrid-interaction honeynet has the ability to manage the level of adversary takeover risk they are willing to assume. By substituting high-interaction honeypots with low-interaction ones, they can effectively reduce the takeover risk. Our goal is to implement these functionalities into our framework.

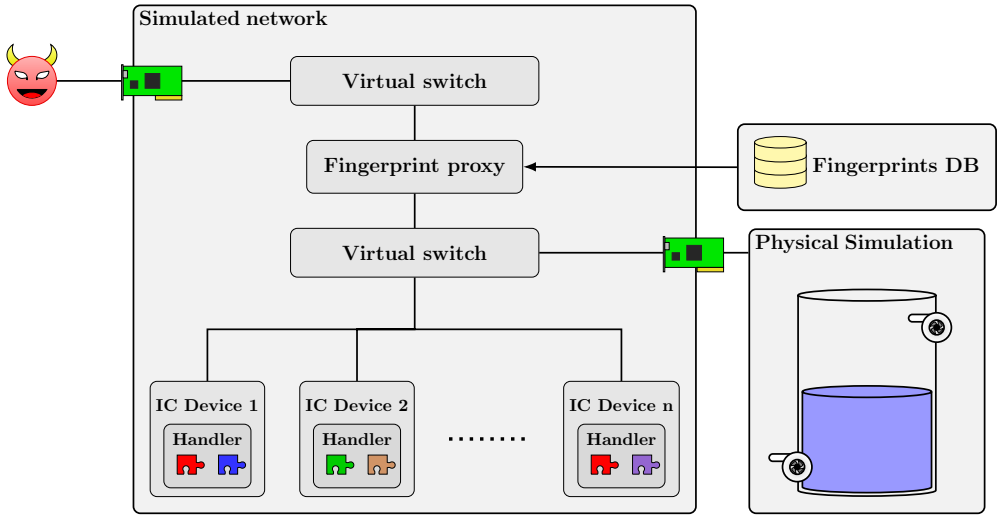
9.2 Framework enhancements

The principal objective of our enhancement, as illustrated in Figure 9.1a, is to provide the attacker with a comprehensive virtual network accessible through a single network interface (as denoted by "1"). We deliberately refrain from making assumptions about the characteristics of the communication channel between this network interface and the attacker. Within this network, the attacker should have the capability to engage with any accessible simulation nodes, and the quality of this interaction must be sufficiently realistic to convince the attacker that it mirrors a legitimate industrial control network.

With the aim of achieving this objective, we establish a link between the network interface of a simulation host, accessible to the attacker, and a virtual switch (indicated as "2"). This linkage provides the attacker with the ability to engage with any simulated hosts connected to this virtual switch. Each of these simulated hosts functions as a representation of a node with a specific role in the



(a) Device mapping modules.



(b) Simulation with device profiles.

Figure 9.1: Framework enhancements toward honeypot functionalities.

simulation, such as a Programmable Logic Controller (PLC), a Human-Machine Interface (HMI), a node simulating a physical process, or any other pertinent node within the simulated scenario.

The central design concept of our enhancement prioritizes adaptability. Each simulated node generates a device handler, which oversees the device module and the industrial protocol listeners (3). The device module encompasses specific con-

trol logic for the scenario, a memory mapping that imitates an actual device, and a synchronization mechanism utilized to interact with the physical process simulation (4). The handler serves as a bridge, connecting the device simulation with the industrial control protocol modules and enabling any actor within the same network to engage with the simulated device through the appropriate protocols (5).

These recent improvements have expanded the capabilities of simulating diverse industrial control scenarios. In each scenario, a configuration file is utilized to define the network structure, device specifications, and the necessary modules. For every device, it is essential to outline 1) network connections, 2) virtual device details (such as manufacturer, model, and device name), 3) the specific industrial control module to be activated, and 4) the industrial control protocols to be accommodated. However, one limitation of this approach is that in the event of a malicious actor conducting reconnaissance activities on the simulated network, they might still be able to identify the simulated devices as the host operating system.

One significant limitation arises from the current OS detection techniques. When using network scanners, they scrutinize the comprehensive network behavior of a specific device to pinpoint particular indicators that suggest a specific operating system. However, an issue often arises with honeypots, as they inadvertently reveal the characteristics of the genuine operating system instead of those intended for the emulated device.

In order to outsmart an attacker and bypass any detection methods that are used to identify the operating system being run, we need to start by creating a custom network profile for each device we want to mimic. There are two key elements to these device profiles: the open ports and their associated protocols,

such as the TCP port 80 for HTTP, and the overall network behavior of the device, including TCP options. To achieve this, we use a fingerprint proxy (refer to Figure 9.1b) to showcase these aspects of the simulated devices to the attacker.

This specialized proxy is designed to precisely emulate the network behavior of a specific device, provided its fingerprint is stored in the database. It serves as a protective barrier, shielding the simulated devices from potential attacks. Additionally, we configure the proxy to intelligently reroute incoming connections to designated industrial control ports, ensuring seamless interaction with the simulated physical processes.

9.3 Physical process simulation and control

For simulating the physical process, we provide two alternative mechanisms. The first option involves utilizing a virtual device and configuring the virtual network topology to ensure that this particular node is completely isolated from any potential interactions with attackers. The second option relies on an external simulator and entails isolating a secondary network interface within the host, serving as the simulation node, with dedicated access to the external simulator.

Now that we have integrated a fingerprint proxy, we have the option to introduce a physical simulation node into the network. Alternatively, we could establish a connection for a secondary network interface to the internal switch, effectively preventing the attacker from gaining direct access.

Factory I/O serves as an external simulator, designed as a Next Generation PLC trainer, to offer industrial scenarios and customizable features. It replicates physical processes in what they refer to as a “scene”, and grants access to virtual sensors and actuators that actively impact the simulation. Our interface with this simulator leverages its built-in Modbus support to retrieve or modify sensor and

actuator values.

We have developed a configurable simulated Programmable Logic Controller (PLC) to manage various scenarios. This simulated PLC communicates with Factory I/O using the Modbus protocol. It continuously monitors the current status of the simulation and executes appropriate control logic based on the simulated system's physical characteristics. The device maintains local state in a simulated memory mapping designed to align with standard industrial control protocols. This mapping consists of four 64K addressable memory blocks, each containing pairs of boolean and 16-bit data values organized in distinct read-only and read-write blocks. This setup allows for the storage of 64K read-only boolean values, 64K read-write boolean values, 64K read-only 16-bit data, and 64K read-write 16-bit data.

We have designed this memory mapping to seamlessly use standard industrial control protocols. For example, if the PLC we want to simulate uses Modbus, then the memory mapping will correspond to Modbus' Discrete Inputs (boolean read-only), Coils (boolean read-write), Input Registers (16-bit read-only), and Holding Registers (16-bit read-write). This mapping is compatible with IEC-104, but the difference is that instead of memory addresses, IEC-104 uses Information Object Addresses (IOA). In this case, the device determines the specific memory mapping it needs to access by the given IEC-104 message and IOA. As an example, the "Single/Double command" in IEC-104 would access boolean read-write memory regions, while "Single-point information" would access either the read-only or read-write boolean mappings.

In conjunction with the physical simulation, we have introduced an additional element to the network in the form of an HMI (Human-Machine Interface) tailored to each simulated scenario. FUXA, an open-source HMI and SCADA dashboard,

is employed to deploy these HMIs. This HMI functions as a point of communication with our simulated devices, mimicking its interaction with genuine industrial control devices. With the augmented capabilities of the HMI, each scenario offers a comprehensive overview of the simulated process, furnishing the attacker with an array of interaction options.

9.4 Honeypot evaluation

Our goal was to compare the effectiveness of a simulated scenario with actual devices in a physical testbed. We gathered fingerprints from real industrial control devices and compared the detection accuracy of network scanners between the real and simulated devices. The output of each scan includes a predicted Operating System and a confidence percentage. Since multiple scans were performed on each device, we also analyzed the frequency of each pair (OS, Confidence) for both the real devices and the simulations. Our objective was to assess the accuracy of standard scanning tools in identifying ICS devices through our own fingerprint extraction. To achieve this, we used the frequency of correct OS detection per device and normalized the results.

Table 9.1 summarizes our results. OS detection performs well for the actual scanned devices, achieving high percentages in seven of the nine ICS equipment evaluated; furthermore, performance improves for our framework, getting over 80% OS prediction confidence for eight of the nine devices. Our approach identifies Industrial Devices as frequently as a person with access to the actual device would find for all the studied real scanned ICS devices, except Allen-Bradley ENBT/A.

There are two real devices with low detection values: Siemens 200sp and Allen-Bradley Micrologix 1400. Similarly, our simulated Allen-Bradley ENBT/A scan identification results are below 50%. In an ideal scenario, scan fingerprinting

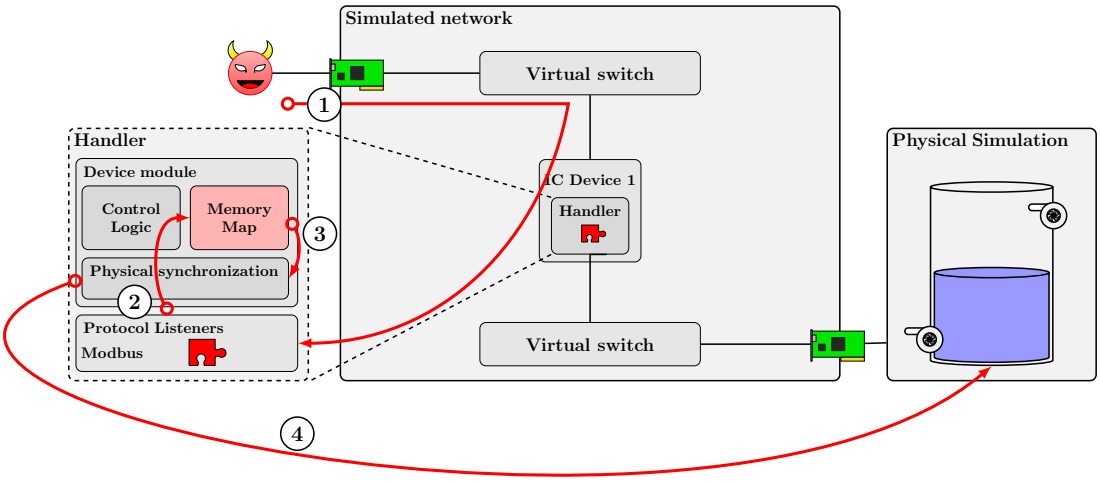
Table 9.1: OS Detection confidence comparison

Device	Real	Simulation
Allen-Bradley ENBT/A	100	40
Micrologix 1400	36	100
Mguard RS4004	100	100
MOXA EDS-405A	86	100
NI-Crio-9024	100	100
NI-Crio-9068	100	100
Siemens 200sp	10	80
Siemens S7-1500	100	100
Siemens S7-1200	100	100

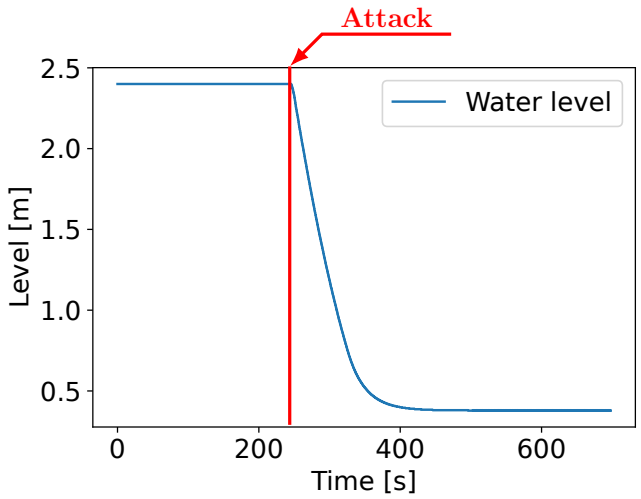
would allow for exact host identification. However, in practice, there are multiple fingerprints that map to a single device, or a single fingerprint that maps to numerous operating systems. Our main hypothesis for the low identification of Siemens 200sp, Micrologix 1400, and virtual Allen-Bradley ENBT/A is that the values found in our extracted fingerprints are very similar to those of other OS. We are encountering the scenario where one fingerprint maps to multiple devices, causing scanners to place high bets on different devices given the conflicting fingerprints. Embedded systems are especially prone to this type of error since third-party vendors often sell their system architecture and firmware to various companies.

As for the physical simulation, we configured our framework with an appropriate network topology to follow the defined threat model, in which the attacker can communicate with the target PLC devices via the fingerprint proxy. This configuration lets the attacker interact with the simulated PLC devices as if they were in the same virtual switch (Figure 9.2a).

Since the attacker can communicate with the simulated devices, they have to issue an appropriate command to modify the physical synchronization component, constantly assert the sensor data from the physical simulation, updating the



(a) Framework interactions with the attacker.



(b) Measured water level.

Figure 9.2: Water tank attack.

memory map sensor values. The new actuator values from the memory map are updated to the simulated holding register, where the PLC stores the target value.

From the viewpoint of the framework's implementation, the protocol listener handles the Modbus connection, accepts the command if the address is within the read-write memory mapping, and modifies the value as requested. Concurrently, the control logic constantly runs a control loop that reads the sensor data, runs the

control logic, and adjusts the actuator values as needed. In addition, the physical synchronization component constantly reads the sensor data from the physical simulation, updates the memory map sensor values, asserts the new actuator values from the memory map, and updates them within the physical simulation.

Figure 9.2a shows the attack flow. ① The attacker sends the Modbus command to the simulated PLC. ② The Protocol Listener receives the command, updating the memory map and forcing the control logic to make the necessary changes. ③ The Physical Synchronization component reads the new actuator values due to the memory map updates and, subsequently, ④ sends them to the physical simulation, which is ultimately affected by the attack.

To corroborate whether the attack was successful, the attacker must acquire some relevant sensor data from a physical source that should be affected by the intended attack. Since sensor data in a PLC is read-only, the attacker must send a command to retrieve this information from either a discrete input, in the case of boolean sensors, or an input register. Since the corresponding sensor values in our chosen scenarios are not discrete, the attacker sends the Modbus command *Read input register* targeting the sensor address that stores the relevant value. In this case, the Protocol Listener accepts the command and returns the current values from the memory map to the attacker.

Our attack aims to disrupt the process that requires a specific water level in the tank by attempting to empty the tank. The attacker must send a zero value to the holding register that stores the set point to accomplish this goal. After executing the attack as described above, we observed the behavior of the water level in the tank, shown in Figure 9.2b. Before the attack, the control logic was kept stable, as the process required. When the malicious command modifies the Set Point in the memory map, the level decreases to a value close to zero, disrupting the

process.

In summary, our high-fidelity physical process integration with Factory I/O allows an attacker to see the changes to the physical system under control, providing a comprehensive hybrid-interaction honeypot. Our modular design can enable future researchers to extend our simulations to several other physical processes. Factory I/O allows designers to create essentially digital twins based on the sensors and actuators it supports.

Chapter 10

Discussion and Conclusion

10.1 Discussion

10.1.1 Lessons learned

In this study, we wanted to evaluate different mechanisms researchers can use for a variety of purposes involving cyber-physical systems. Testing theoretical attacks, countermeasures, or even deploying an existing malware. These tasks can be dangerous in many scenarios involving this kind of system. Unsurprisingly, the requirements and restrictions for the simulations rise from the intended purpose.

After conducting the various evaluations of the main scenarios we wanted to study, there is a pattern for this kind of research. As an overall methodology, we identify the following steps when dealing with any virtualization attempt:

1. Clarify the simulation objective. While there are several simulation mechanisms for any given system, the goal of the simulation might not be appropriate for the existing environment. With a clear understanding of the goal, one can ascertain whether a specific tool or module is useful to conducting the virtualization. For instance, several of the existing simulation

environments prior to this work were not suitable for studying the malware samples.

2. Evaluate the constraints and restrictions imposed by both the real system and the goal. The simulated environment must accommodate any specific restrictions associated with the intent. As an example, the malware samples we studied required specific conditions and network protocols to properly display their behavior.

3. Isolate the environment. Needless to say, when conducting security research in a virtualized environment, particularly when handling malware, it is essential to avoid exposing any other network to potentially malicious or hazardous network traffic.

4. Identify anomalies. Any anomalous behavior observed during the execution of the virtualized scenario provides additional insights to refine the virtualization, enhancing its capabilities.

With our study, we learned that using virtualized systems is not only a viable way to replicate cyber-physical systems, but also a recommended practice. We were able to build reasonable abstractions of real world systems in various virtualized environments, contrasting their behavior with their physical counterparts, determining the feasibility of using such systems as research tools. Both attack and defense stratagems can be studied with this approach.

The primary and fundamental aspect of the simulation that represented most of the success was one: reliable and compliant protocol modules. Cyber-physical systems use a plethora of network protocols, some of which are proprietary. A faithful reproduction of such protocols proved to be instrumental to the success of this study.

10.1.2 Safety, security, and other ethical concerns

Throughout this study we worked both with physical devices and known malicious software. The very nature of our study revolves around mechanisms to safely test ideas and scenarios involving hazardous conditions of physical systems, and proper solutions to effectively isolate any malicious execution from the world.

With regards to the drone attacks. All the real-world tests we performed were under strictly controlled conditions. The drone was never flown beyond two meters above ground, and any attacks limited the horizontal speed of the drone to half a meter per second. These constraints prevented any harmful consequences while testing our attacks against the device. As for the specific GPS attacks, we abstained from testing these attacks in the real world, for we had no way to restrict the attack to specifically target the drone, showcasing the importance of a simulation environment.

Any proof of concept we developed involving malicious actions were only executed in the isolated environment and no public disclosure of these artifacts was made to avoid potential malfeasance by third-parties. The attacks were purely theoretical in nature, but can represent a real threat to the targeted systems. Thus, we only evaluated the feasibility in the virtualized scenario.

Finally, the entire analysis conducted over the malware samples was carefully controlled, avoiding the dissemination of the malicious artifact, and preventing any malicious network traffic from ever leaving the virtualized environment. We also need to clarify that one of the malware samples has been made publicly available by a third party (The Industroyer 2 sample.)

10.2 Conclusion

In this study we showcased several ways in which researchers can leverage the advantages of virtualized systems to conduct cyber-security research over cyber-physical systems. In particular, we develop an extensible emulation framework realistic enough to fool a real-world malware sample. We also present the results of the first in-depth analysis of a known malware developed by a nation-state actor, showcasing previously unknown insights revealed by our unique approach.

By leveraging virtualized environments, we were able to test several approaches to cyber-security research. From new conceptual attacks to potential defense mechanisms, virtualized systems are a practical tool to evaluate these approaches without risking the operational integrity of the physical processes involving the studied cyber-physical systems.

Bibliography

[1] Hossein Ghassempour Aghamolki, Zhixin Miao, and Lingling Fan. A hardware-in-the-loop scada testbed. In *2015 North American Power Symposium (NAPS)*, pages 1–6, 2015.

[2] Cristina Alcaraz and Sherali Zeadally. Critical infrastructure protection: requirements and challenges for the 21st century. *International journal of critical infrastructure protection*, 8:53–66, 2015.

[3] Daniele Antonioli, Anand Agrawal, and Nils Ole Tippenhauer. Towards high-interaction virtual ICS honeypots-in-a-box. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy (co-located with CCS)*, pages 13–22. ACM, 2016.

[4] Daniele Antonioli and Nils Ole Tippenhauer. MiniCPS. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy - CPS-SPC '15*, pages 91–100, New York, New York, USA, 2015. ACM Press.

[5] Wissam Aoudi, Mikel Iturbe, and Magnus Almgren. Truth will out: Departure-based process-level detection of stealthy attacks on control systems. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 817–831. ACM, 2018.

[6] David C. Bergman, Dong Jin, David M. Nicol, and Tim Yardley. The virtual power system testbed and inter-testbed integration, 2009.

[7] Alvaro A Cardenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the ACM symposium on information, computer and communications security*, pages 355–366, 2011.

[8] Anton Cherepanov. Win32/industroyer a new threat for industrial control systems, 6 2017.

[9] Gary Cohen. Throwback attack: An insider releases 265,000 gallons of sewage on the maroochy shire. `https://www.industrialcybersecuritypulse.com/facilities/throwback-attack-an-insider-releases-265000-gallons-of-sewage-on-the-maroochy-shire/`.

[10] Drew Davidson, Hao Wu, Robert Jellinek, Thomas Ristenpart, and Vikas Singh. Controlling uavs with sensor input spoofing attacks. In *Proceedings of the 10th USENIX Conference on Offensive Technologies*, WOOT'16, pages 221–231, Berkeley, CA, USA, 2016. USENIX Association.

[11] Dragos, Inc. Crashoverride: Analysis of the threat to electric grid operations, 6 2017.

[12] ESET. Industroyer2: Industroyer reloaded, 4 2022.

[13] Wenjun Fan, Zhihui Du, and David Fernández. Taxonomy of honeynet solutions. In *2015 SAI Intelligent Systems Conference (IntelliSys)*, pages 1002–1009. IEEE, 2015.

[14] OPC Foundation. What is opc? `https://opcfoundation.org/about/what-is-opc/`, 2017.

[15] The Open Networking Foundation. Openflow switch specification version 1.5.1 (protocol version 0x06). `http://www.opennetworking.org`, 2015.

[16] Lorenzo Franceschi-Bicchierai. Hackers shut down heating in ukrainian city with malware, researchers say. `https://techcrunch.com/2024/07/23/hackers-shut-down-heating-in-ukrainian-city-with-malware-researchers-say/`.

[17] Javier Franco, Ahmet Aris, Berk Canberk, and A Selcuk Uluagac. A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems. *IEEE Communications Surveys & Tutorials*, 23(4):2351–2383, 2021.

[18] Kevin Fu and Wenyuan Xu. Risks of trusting the physics of sensors. *Communications of the ACM*, 61(2):20–23, 2018.

[19] Jairo Giraldo, Esha Sarkar, Alvaro A. Cardenas, Michail Maniatakos, and Murat Kantarcioglu. Security and Privacy in Cyber-Physical Systems: A Survey of Surveys. *IEEE Design & Test*, 34(4):7–17, aug 2017.

[20] Jairo Giraldo, David Urbina, Alvaro Cardenas, Junia Valente, Mustafa Faisal, Justin Ruths, Nils Ole Tippenhauer, Henrik Sandberg, and Richard Candell. A survey of physics-based attack detection in cyber-physical systems. *ACM Computing Surveys (CSUR)*, 51(4):76, 2018.

[21] Andy Greenberg. How russia-linked malware cut heat to 600 ukrainian buildings in deep winter. `https://www.wired.com/story/russia-ukraine-frostygoop-malware-heating-utility/`.

[22] Andy Greenberg. Crash Override: The malware that took down a power grid, 6 2017.

[23] Ilya Grinberg, Matin Meskin, and M. Safiuddin. Test bed for a cyber-physical system (cps) based on integration of advanced power laboratory and extensible messaging and presence protocol (xmpp). In *2015 ASEE Annual Conference Exposition*, 06 2015.

[24] Dina Hadžiosmanović, Robin Sommer, Emmanuele Zambon, and Pieter H Hartel. Through the eye of the PLC: semantic security monitoring for industrial processes. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, pages 126–135. ACM, 2014.

[25] Adam Hahn, Aditya Ashok, Siddharth Sridhar, and Manimaran Govindarasu. Cyber-physical security testbeds: Architecture, application, and evaluation for smart grid. *IEEE Transactions on Smart Grid*, 4(2):847–855, 2013.

[26] Todd E Humphreys, Brent M Ledvina, Mark L Psiaki, Brady W O'Hanlon, and Paul M Kintner. Assessing the spoofing threat: Development of a portable gps civilian spoofer. In *Radionavigation Laboratory Conference Proceedings*, 2008.

[27] Celine Irvene, David Formby, Samuel Litchfield, and Raheem Beyah. HoneyBot: A honeypot for robotic systems. *Proceedings of the IEEE*, 106(1):61–70, jan 2018.

[28] Darryl Jenkins and Bijan Vasigh. *The economic impact of unmanned aircraft systems integration in the United States*. Association for Unmanned Vehicle Systems International (AUVSI), 2013.

[29] Andrew J Kerns, Daniel P Shepard, Jahshan A Bhatti, and Todd E Humphreys. Unmanned aircraft capture and control via gps spoofing. *Journal of Field Robotics*, 31(4):617–636, 2014.

[30] Amit Kleinmann, Ori Amichay, Avishai Wool, David Tenenbaum, Ofer Bar, and Leonid Lev. Stealthy deception attacks against scada systems. In *Computer Security*, pages 93–109. Springer, 2017.

[31] Denis Foo Kune, John Backes, Shane S Clark, Daniel Kramer, Matthew Reynolds, Kevin Fu, Yongdae Kim, and Wenyuan Xu. Ghost talk: Mitigating emi signal injection attacks against analog sensors. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 145–159. IEEE, 2013.

[32] Nozomi Networks Labs. Industroyer2: Nozomi networks labs analyzes the iec 104 payload, 4 2022.

[33] Vedere Labs. Industroyer2 and incontroller: In-depth technical analysis of the most recent ics-specific malware, 7 2022.

[34] Robert M Lee, Michael J Assante, and Tim Conway. Ics defense use case 5: Analysis of the cyber attack on the ukrainian power grid. Technical report, SANS Industrial Control Systems, Mar 2016.

[35] Max Lesser, Simon Fellows, and Oakley Cox. Defending operational technology (OT) in kinetic, cyber, and hybrid warfare.

[36] Samuel Litchfield, David Formby, Jonathan Rogers, Sakis Meliopoulos, and Raheem Beyah. Rethinking the Honeypot for Cyber-Physical Systems. *IEEE Internet Computing*, 20(5):9–17, sep 2016.

[37] Yao Liu, Peng Ning, and Michael K Reiter. False data injection attacks against state estimation in electric power grids. In *Proceedings of the conference on Computer and communications security (CCS)*, pages 21–32. ACM, 2009.

[38] Efrén López-Morales, Carlos Rubio-Medrano, Adam Doupé, Yan Shoshitaishvili, Ruoyu Wang, Tiffany Bao, and Gail-Joon Ahn. Honeyplc: A next-generation honeypot for industrial control systems. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 279–291, Virtual Event, USA, 2020. Association for Computing Machinery.

[39] Malaz Mallouhi, Youssif Al-Nashif, Don Cox, Tejaswini Chadaga, and Salim Hariri. A testbed for analyzing security of scada control systems (tasscs). In *ISGT 2011*, pages 1–7, 2011.

[40] Dan Maloney. Black starts: How the grid gets restarted. `https://hackaday.com/2021/07/15/black-starts-how-the-grid-gets-restarted/`.

[41] Michael J. McDonald, Gregory N. Conrad, Travis C. Service, and Regis H. Cassidy. Cyber effects analysis using vcse promoting control system reliability, 2008.

[42] Thomas Morris, Anurag Srivastava, Bradley Reaves, Wei Gao, Kalyan Pavurapu, and Ram Reddi. A control system testbed to validate critical infrastructure protection concepts. *International Journal of Critical Infrastructure Protection*, 4(2):88–103, 2011.

[43] Luciana Obregon and Barbara Filkins. Secure Architecture for Industrial Control Systems Secure Architecture for Industrial Control Systems Secure Architecture for Industrial Control Systems 2. Technical report, SANS Institute, 2015.

[44] Donghui Park, Julia Summers, and Michael Walstrom. Cyberattack on critical infrastructure: Russia and the ukrainian power grid attacks. https://jsis.washington.edu/news/cyberattack-critical-infrastructure-russia-ukrainian-power-grid-attacks/, Jul 2019.

[45] Andrés F. Murillo Piedrahita, Vikram Gaur, Jairo Giraldo, Alvaro A. Cardenas, and Sandra Julieta Rueda. Virtual incident response functions in control systems. *Computer Networks*, 135:147–159, Apr 2018.

[46] Pavel Polityuk, Oleg Vukmanovic, and Stephen Jewkes. Ukraine's power outage was a cyber attack: Ukrenergo, 1 2017.

[47] Venkat Pothamsetty and Matthew Franz. SCADA HoneyNet Project: Building Honeypots for Industrial Networks. http://scadahoneynet.sourceforge.net/, 2005.

[48] PX4 Autopilot Project. Px4 architecture overview. https://dev.px4.io/en/concept/architecture.html.

[49] PX4 Autopilot Project. Px4/jmavsim: Simple multirotor simulator with mavlink protocol support. https://github.com/PX4/jMAVSim.

[50] The Dronecode Project. Mavlink developer guide. https://mavlink.io/en/.

[51] Ken Proska, John Wolfram, Jared Wilson, Dan Black, Keith Lunden, Daniel Kapellmann, Nathan Brubaker, Tyler McLellan, and Chris Sistrunk. Sandworm disrupts power in Ukraine using a novel attack against operational technology, Nov 2023.

[52] Adam Rawnsley. Iran's alleged drone hack: Tough, but possible. Wired, 2011.

[53] Juan Enrique Rubio, Cristina Alcaraz, Rodrigo Roman, and Javier Lopez. Analysis of Intrusion Detection Systems in Industrial Ecosystems. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications*, volume 4, pages 116–128, Madrid, Spain, aug 2017. SCITEPRESS - Science and Technology Publications.

[54] Vahid Salehi, Ahmed Mohamed, Ali Mazloomzadeh, and Osama A. Mohammed. Laboratory-based smart power system, part i: Design and system development. *IEEE Transactions on Smart Grid*, 3(3):1394–1404, 2012.

[55] Frank Schroth. Gartner predicts 3 million drones to be shipped in 2017. https://dronelife.com/2017/02/10/gartner-predicts-3-million-drones-shipped-2017/, 2017.

[56] Steffen Schütte., Stefan Scherfke., and Michael Sonnenschein. Mosaik - smart grid simulation api - toward a semantic based standard for interchanging smart grid simulations. In *Proceedings of the 1st International Conference on Smart Grids and Green IT Systems - Volume 1: SMARTGREENS,*, pages 14–24, Porto, Portugal, 2012. INSTICC, SciTePress.

[57] Mohammad Shahidehpour and Mohammad Khodayar. Cutting campus energy costs with hierarchical control: The economical and reliable operation of a microgrid. *IEEE Electrification Magazine*, 1(1):40–56, 2013.

[58] Richard William Skowyra, Andrei Lapets, Azer Bestavros, and Assaf Kfoury. Verifiably-safe software-defined networks for CPS. In *Proceedings of the 2nd ACM international conference on High confidence networked systems - HiCoNS '13*, page 101, New York, New York, USA, 2013. ACM Press.

[59] Joe Slowik. Anatomy of an attack: detecting and defeating crashoverride, 2018.

[60] Joe Slowik. Crashoverride: Reassessing the 2016 Ukraine electric power event as a protection-focused attack, 2019.

[61] Yun Mok Son, Ho Cheol Shin, Dong Kwan Kim, Young Seok Park, Ju Hwan Noh, Ki Bum Choi, Jung Woo Choi, and Yong Dae Kim. Rocking drones with intentional sound noise on gyroscopic sensors. In *24th USENIX Security symposium*. USENIX Association, 2015.

[62] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. Interaction matters: A comprehensive analysis and a dataset of hybrid iot/ot honeypots. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 742–755, Austin, TX, USA, 2022. Association for Computing Machinery.

[63] Alexandru L Stancu, Simona Halunga, Alexandru Vulpe, George Suciu, Octavian Fratu, and Eduard C Popovici. A comparison between several Software Defined Networking controllers. In *2015 12th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, TELSIKS 2015*, pages 223–226, Nis, Serbia, oct 2015. IEEE.

[64] Mininet Team. Mininet. `http://mininet.org/`, 2017.

[65] Nils Ole Tippenhauer, Christina Pöpper, Kasper Bonne Rasmussen, and Srdjan Capkun. On the requirements for successful gps spoofing attacks. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 75–86. ACM, 2011.

[66] Timothy Trippel, Ofir Weisse, Wenyuan Xu, Peter Honeyman, and Kevin Fu. Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 3–18. IEEE, 2017.

[67] David I Urbina, Jairo A Giraldo, Alvaro A Cardenas, Nils Ole Tippenhauer, Junia Valente, Mustafa Faisal, Justin Ruths, Richard Candell, and Henrik Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1092–1105. ACM, 2016.

[68] Gérard Wagener, Radu State, Thomas Engel, and Alexandre Dulaunoy. Adaptive and self-configurable honeypots. In *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pages 345–352, 2011.

[69] Yong Wang, Zhaoyan Xu, Jialong Zhang, Lei Xu, Haopei Wang, and Guofei Gu. SRID: State relation based intrusion detection for false data injection attacks in SCADA. In *Proceedings of European Symposium on Research in Computer Security (ESORICS)*, pages 401–418. Springer, 2014.

[70] Elizabeth Weise. Mysterious gps glitch telling ships they're parked at airport may be anti-drone measure. USA Today, Sep 2017.

[71] Changhoon Yoon, Taejune Park, Seungsoo Lee, Heedo Kang, Seungwon Shin, and Zonghua Zhang. Enabling security functions with SDN: A feasibility study. *Computer Networks*, 85:19–35, jul 2015.

[72] Daniel Zafra, Raymond Leong, Chris Sistrunk, Ken Proska, Corey Hildebrant, Keith Lunden, and Nathan Brubaker. Industroyer.v2: Old malware learns new tricks, 4 2022.

[73] Mark Zeller. Myth or reality—does the aurora vulnerability pose a risk to my generator? In *2011 64th Annual Conference for Protective Relay Engineers*, pages 130–136. IEEE, 2011.

[74] Kim Zetter. Inside the cunning, unprecedented hack of Ukraine's power grid, 3 2016.