University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Commercial off-the-shelf Operating Systems in Industrial Control Systems: A vulnerability assessment

Karnbongkot Boonriong

March 24, 2023

Abstract

Industrial Control Systems (ICS) are becoming increasingly interconnected with the integration of Information Technology. Devices with commercial-off-the-shelf operating systems are being used in ICS for their cost-efficient nature and IT capabilities. This modernisation, although beneficial, exposes ICS to a wider attack surface. ICS researchers need a safe environment to perform assessments of these attacks. Therefore, this work presents a virtual testbed to assess the impact of the attack designed in this theme. Furthermore, we propose a method for attack detection over the network as an option for intrusion detection in an ICS environment.

We built the virtual testbed using virtual machine and container technologies, combined with simulation tools. The attack is designed according to the MITRE ATT&CK Framework for generalisability and is performed over the testbed. The virtual testbed is capable of simulating ICS operation and exhibits a change of behaviour during the attack. The attack was successful but with flaws due to the lack of consideration at the physical process level.

We profile the network traffic using Shannon's entropy on the network feature distribution. This reveals the characteristics of the network during each step of the attack. We compare the performance of five machine learning models for anomaly detection on the data, where we find Isolation Forest to be the model with the highest performance. However, we face difficulties in malware-generated traffic detection due to its similar characteristics to normal traffic.

Education Use Consent

I hereby grant my permission for this project to be stored, distributed and shown to other University of Glasgow students and staff for educational purposes. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Signature: Karnbongkot Boonriong Date: 24 March 2023

Acknowledgement

Firstly, I would like to thank my supervisor, Dr Angelos K. Marnerides, for his endless support. His suggestions and guidance have made a significant improvement to the project. I would also like to thank my parents for their sacrifices which made this work possible. Finally, I would like to thank my friends for their encouragement and mental support throughout the course of this project.

Contents

1 Introduction

1.1 Motivation

Operational Technology (OT) systems such as Industrial Control Systems (ICS) are becoming much more intertwined with Information Technology (IT) systems. The integration of IT devices using Commercial-Off-The-Shelf (COTS) Operating Systems brought common IT vulnerabilities to ICS. The Common Vulnerabilities and Exposures (CVE) score assigned to IT threats is generally the indicator of vulnerability severity. However, the level of impact these threats have on ICS is not equal to the common IT systems and is different for every ICS because of its associated physical process. Therefore, security researchers must find a way to assess the impact of IT threats on ICS without causing harm to the system.

ICS security research is commonly performed on a lab-scale ICS. While this produces results fidelity to the real-world ICS, it can be costly and unscalable. To optimise for the constraints, many researchers developed virtual testbeds and frameworks for ICS security research, aided by developing ICS component simulators. Researchers can perform different types of attacks on the testbed and observe the impact to assess the threat severity.

Intrusion detection systems (IDS) are deployed in the ICS network for the detection of anomalous network states. Anomaly-based IDS is the most commonly used type of IDS, due to its ability to detect both known and unknown anomalous states. Network anomaly detection is therefore a crucial field of research for ICS security.

1.2 Aims and Objectives

1. **Design and develop a virtual testbed for ICS attack simulation.**
   - Identify the virtual testbed requirements.
   - Build the virtual testbed using established tools.
   - Evaluate the virtual testbed by its ability to exhibit the impact of the attacks.
2. **Implement a full chained attack based on MITRE ATT&CK Framework.**
   - Identify the threat model for the virtual testbed.
   - Implement the attack based on the threat model.
   - Evaluate the effectiveness of the attack by analysing its impact on the virtual testbed, focusing on the ICS.
   - Evaluate the success of the attack chain by comparing the outcome of each tactic to its intended outcome according to the MITRE ATT&CK Framework.
3. **Produce methods for attack detection.**
   - Collect network data on the virtual testbed during normal operation and during the attack.
   - Profile the network's normal behaviour using collected data.
   - Implement detection methods using different machine learning algorithms for anomaly detection.
   - Evaluate the effectiveness of the methods using machine learning performance metrics.

2 Background and Related Work

2.1 Industrial Control System

Industrial Control System is a generalised term for several types of control systems typically consisting of control loops, human interfaces, and network protocols. ICSs are used to operate and automate physical processes such as water management and distribution systems. The architecture of ICSs can be visualised using the Purdue Enterprise Reference Architecture (Williams 1994) in Table 2.1. Note that the modern ICSs do not strictly follow the Purdue Model, and the separation of levels and functionality is less defined.

Table 2.1: *The Purdue Enterprise Reference Architecture*

Purdue Level	Description	Example
Level 0	Field Devices for sensing and manipulating the physical process	- Sensors - Actuators - IIoT Devices
Level 1	Local controller for providing automated control of the physical process	- Programmable Logic Controllers (PLCs) - Control processors - Programmable relays - Remote terminal units (RTUs)
Level 2	Control systems supervisory for monitoring and controlling the physical process	- Human-machine Interface (HMI) - Engineering workstation - SCADA Softwares
Level 3	Site-wide supervisory for monitoring and maintaining production performance.	- Management servers - Analytic systems - Historians
DMZ		
Level 4	Business Networks	- Email Servers - Business Workstations - File servers
Level 5	Enterprise Networks	- Data Centers - Enterprise Active Directory (AD) - Enterprise Security Operations Centre (SOC)

Today ICSs are often delivered as a complex infrastructure of IT solutions including commercial off-the-shelf technologies (Holm et al. 2015). The main reasons behind the use of COTS technology are increased functionality, effectiveness, and lower costs. COTS technology has also introduced common IT issues, such as malware, into ICS. While short-term unavailability is acceptable in most IT systems, it is unacceptable in ICS and could cause severe damage to the system. Careful assessments and considerations must be taken in the design of ICS with COTS technology to evaluate the vulnerability and risks of using COTS.

2.2 Programmable Logic Controller

Programmable logic controllers (PLC) operate in level 1 of the Purdue reference architecture. PLCs are used to automate the physical process, generally by receiving input signals from the sensors and sending output signals to the actuators. The PLC operates in an infinite loop of scanning for inputs, executing its program, and writing outputs. The PLC is programmed using PLC programming languages such as Ladder Logic, Function Block Diagram, and Structured text. Most PLC programming languages comply with the IEC 61131-3 standard. (Tiegelkamp and John 2010)

The baseline components of a PLC include:

- Processor unit (CPU) for reading input, executing program, and writing output.
- Power supply unit.
- Memory unit for storing input and program data.
- I/O interface for sending and receiving data from other devices. Discrete I/O and Analog I/O are separated.
- Communication unit for communication on the network with other devices.

2.2.1 PLC Communication

Most PLCs can communicate with other devices using common ICS protocols or their proprietary protocol. Some PLC manufacturer has developed the protocol for their PLCs, for example, Siemens developed the S7comm protocol, Allan-Bradley developed the DH+ protocol, Schneider Electric developed the UMAS protocol, and Mitsubishi developed the MELSEC protocol. In this dissertation, we will focus on one of the common ICS protocols, the Modbus protocol.

The Modbus protocol (ModbusOrganization 2012) was developed by Modicon in 1979. It then became the de facto standard communication protocol for industrial electronic devices. Modbus is a request and reply protocol, where the Modbus "slave" can request a service from a Modbus "master" by specifying the predefined function code in the request.

There is no standardised memory addressing scheme for the PLC. Generally, PLC memory is divided into areas based on data type (bit, integer, word, etc.) and functionality (input, output, local, etc.). Modbus protocol models data on a PLC into four categories, listed in Table 2.2. The PLCs that support the Modbus protocol must provide a mapping between the data model and the PLC memory areas.

Table 2.2: *Modbus data model*

Type	Size (Bit)	Permission	Purpose
Discrete Inputs	1	Read-Only	Input
Coils	1	Read-Write	Output
Input Registers	16	Read-Only	Input
Holding Registers	16	Read-Write	General purposes

The Modbus function code was defined based on the data model. There are three types of function codes in Modbus protocol: public, user-defined, and reserved. The public function code ranges from 1 to 64, 73 to 99, and then 111 to 127. The user can define function codes from 65 to 72 and from 100 to 110. The reserved function code is the code currently used by the company or legacy product and is not available to the public. The most basic function codes which will appear later in this paper are:

- Read Discrete Inputs (code 02)
- Read Coils (code 01)

- Write Single Coil (code 05) / Write Multiple Coils (code 15)
- Read Input Registers (code 04)
- Read Holding Registers (code 03)
- Write Single Register (code 06) / Write Multiple Registers (code 16)

By specifying the function code and following the format of the Modbus request, a device such as the engineering workstation can request the PLC to perform a specific function or form a reply with the requested information.

2.2.2 PLC Vulnerabilities

The PLC has been the preferred target of attacks because of its ability to control the physical process and its dependency on COTS operating systems. Some of the most popularly used operating systems in the PLC are VxWorks, Microware OS-9, Linux and Microsoft Windows, each with its own vulnerabilities (Milinković and Lazić 2012). This subsection will discuss some of the PLC vulnerabilities that have been discovered to give an overview of how the PLC can be attacked.

Sandaruwan et al. (2013) shows two generic PLC attacks and 3 Siemens Semantics S7 PLC attacks. The generic PLC attacks are By-pass logic attacks and Brute-force output attacks. By-pass logic attacks exploit the fact that PLC programs store the program variables in the holding registers. These variables, if changed, could affect the output of the PLC. An attacker could gain physical access to the PLC or infect the PLC over the network with a worm that modifies the registers. The brute force output attack takes advantage of one of the standard functionalities of the PLC, forcing output. This functionality allows operators to remotely change the output of the PLC. The attacker could gain network access to the PLC and force change the output to harmful values.

The PLC's program could also be the source of its vulnerability. The following list shows the possible code-level vulnerabilities in the PLC, focusing on Ladder Logic language programs by Serhane et al. (2018).

- Duplicated instruction: Calling certain operations repeatedly may cause undesired effects.
- Snooping: The program can be written to log a parameter for spying purposes without affecting the logic flow and the original purposes.
- Bypassing: Manually forcing the values of certain variables while the ladder logic is online.
- DoS: The attacker can upload a malicious ladder logic program that will be triggered and causes a slow-down or total halting of the PLC.
- Race condition: Multiple codes or operands racing can cause inconsistency in the output.
- Keeping PLC in program mode: Leaving PLC in "unlock" mode or "program" mode allows the attacker to upload malicious code.
- Lack of authentication: If authentication is not needed to upload a new program to the PLC, the attacker will be able to upload vulnerable programs onto the PLC.
- Improperly handling diagnostics and alarms: Not handling faulty conditions properly can lead to errors and harmful states not being controlled before damage is done.

Wardak et al. (2016) tested the strength of the PLC password access control mechanism by launching multiple attacks over the network. They were able to execute a start/stop command on the PLC by launching a TCP replay attack. They also successfully reverse-engineered the password encoded in the PLC communication packets. Using the password acquired, they could perform unauthorised password settings and updates.

In 2021, researchers from Soonchunhyang University, Kaspersky and Claroty discovered **CVE-2021-22681**. This is a vulnerability in Rockwell Automation Logix Controllers which allows

the attacker to discover the key used to verify the communication between the PLC and the workstation. An attacker with the key can spoof the communication and upload malicious code onto the PLC.

In summary, the PLC has vulnerabilities in many of its components. It is vulnerable to OS-level exploits, code-level exploits, and network-level exploits. A compromised PLC will give the attacker access to the physical process, where they can cause damage to the ICS.

2.3 ICS Attacks

Cyber-attacks on ICS became prominent after the discovery of the Stuxnet worm in 2010. Since then, there have been many emerging ICS malware and viruses. The motivations for attacking ICS are often financial gain and political or military objectives. When a commercial off-the-shelf product is integrated into industrial control systems, the ICS inherit COTS vulnerabilities, increasing its vulnerable vector. This gives the advantage to the attacker since COTS vulnerabilities are widely known. This section will explore recent malware used in ICS attacks and the common vulnerability that was weaponised.

2.3.1 Stuxnet

The Stuxnet worm was discovered in 2010. The size of the worm is 500KBytes and it was written in multiple languages. Stuxnet exploits four zero-day vulnerabilities, two of which were not previously disclosed. Unlike most malware, Stuxnet's target is industrial control systems with selective components. The malware consists of a layered attack that targeted three different systems:

- The Windows operating system
- Siemens PCS 7, WinCC and STEP7 industrial software applications that run on Windows
- Siemens S7 PLCs

The antivirus company, Symantec, set up monitoring for the Stuxnet command and control server. Their data showed that there were approximately 100,000 infected hosts, approximately 60% located in Iran (Falliere et al. 2011). This leads to the widely accepted conclusion that Stuxnet's target was Iran's nuclear facilities.

The Stuxnet entry point is assumed to be a removable flash drive since most industrial control systems are behind a firewall. Once the worm is installed in the local network, it looks for vulnerable Windows PC to infect. The targeted PC will be infected using two zero-day vulnerabilities described below.

- **The Print Spooler Service Impersonation Vulnerability (CVE-2010-2729).** When printer sharing is enabled on the target machine, the Windows print spooler service does not properly validate the user's spooler access permissions. This allows remote attackers to create files in a system directory, and execute arbitrary code, by sending a specially crafted print request over a remote procedure call (RPC).
- **The Server Service Vulnerability (CVE-2008-4250).** This vulnerability allows remote attackers to execute arbitrary code by sending a specially crafted RPC request that will cause overflow during path canonicalization.

Another propagation method for Stuxnet is to copy itself onto a removable drive, exploiting the **LNK Vulnerability (CVE-2010-2568)**. ICS operators often exchange data and programs between Windows PC using removable drives, since the network might not connect the source and destination of the data.

In addition to spreading to other Windows PC, Stuxnet also seeks Windows servers that run Siemens Simatic WinCC and PCS 7 software. It exploits the **Hardcoded Password Vulnerability**

(CVE-2010-2772) to access the back-end database and gain privilege. By injecting payload into the files the software uses, the worm could gain access and control of the Siemens S7 PLCs.

Stuxnet reprogrammed PLCs and altered the frequency of electrical current output to the motor. This causes the speed of the target motor to change back and forth between high and low speed, causing damage and sabotaging the process (Farwell and Rohozinski 2011).

2.3.2 BlackEnergy

BlackEnergy malware has been a part of many cyber incidents. One notable ICS attack that utilises BlackEnergy was the Ukraine power grid attack, where the attacker used a variant of BlackEnergy3 to gain initial access and persistence in the targeted system. The attack successfully compromised three energy companies and shut down their stations, which resulted in the customers having no electricity for one to six hours.

The first version of BlackEnergy malware was an HTTP-based botnet for a DDoS attack discovered in 2007 by Arbor Networks. At the time, BlackEnergy 1 made itself undetectable by using a runtime encrypter.

BlackEnergy 2 was discovered in 2010. It has switched from HTTP botnet architecture to modular architecture. The original rootkit from the first version was combined with new functions for unpacking and injecting modules into processes (Stewart 2010). It infects a machine using a "dropper(.exe)" which will decrypts the binary and installs its rootkit as a service. Since it was made in a form of a Windows kernel driver, in some Windows machines, the Microsoft driver signing policy requires kernel-mode drivers to have a valid digital signature. At this stage, some attackers used a DSEFix, a tool that exploits a **vulnerability in the legitimate VirtualBox driver (CVE-2008-3431)** to bypass the signature check (Cherepanov and Lipovsky 2016).

The malware performs privilege escalation by exploiting a **vulnerability in Microsoft kernel (CVE-2008-1084)**, giving itself permission to add new services. The installed rootkit then performs further actions such as API hooking and injecting the main DLL into svchost.exe. This gives the malware more capability than the original version. It has been used for espionage, fraud, stealing user credentials or as a key logger, scanning networks, sending spam and more (Khan et al. 2016).

The third version is a simplification of the previous version. It directly injects its main DLL into the target, rather than using a rootkit component.

BlackEnergy heavily utilises spear-phishing, the attacker often sends emails containing trojan files to the target. When the target opens the file, the trojan will install the malware on the target machine. Below is the list of vulnerabilities that have been used to create the trojan files.

- **TIFF image trojan (CVE-2013-3906).**
- **MSCOMCTL.OCX RCE Vulnerability (CVE-2012-0158).**
- **Windows OLE Remote Code Execution Vulnerability (CVE-2014-4114).**

2.4 Network Anomaly Detection

Network anomaly detection is the process of monitoring the network to detect anomaly behaviour which might be caused by misuse, failure, or malicious attacks. The steps to anomaly detection can be simplified to network sampling, sample profiling, and detection or classification. Different methods for the execution of each step are chosen based on their compatibility with the network. This section will outline some of the methods used in profiling and detection, and describe in detail the detection models used in this dissertation.

2.4.1 Profiling and Detection Methods

Fernandes et al. (2019) classified anomaly detection methods and techniques into seven categories, described below.

Statistical Methods are commonly based on a probabilistic model associated with the network data. A challenge in this method is finding a numerical threshold for determining outliers. The common techniques used in statistical profiling and detection are:

- Wavelets Analysis (Callegari et al. 2011; Hamdi and Boudriga 2007)
- Principal Component Analysis (Lakhina et al. 2004; O'Reilly et al. 2016)
- Covariance metric (Xie et al. 2014; Huang et al. 2016)

Clustering Methods rely on partitioning samples into groups containing similar samples. The outlier sample tends to have a larger distance from the common clusters. Clustering algorithms can be used to detect anomaly (Rajasegarar et al. 2014; Mazel et al. 2011), or be used to profile the data before passing it onto other algorithms (Karami and Guerrero-Zapata 2015; He et al. 2017).

Finite State Machine (FSM) or finite automata, is a mathematical model consisting of states, transitions and actions. It can be used to represent a computer problem. The machine allows the exploration of every possible state of a system. Example works where FSM is used are Estevez-Tapiador et al. (2003), Su (2010), and Hammerschmidt et al. (2016).

Classification-based Methods build classifiers which will be used to classify samples into normal and anomalous states. Popularly used classifiers are the followings:

- Naïve Bayesian (Swarnkar and Hubballi 2016)
- Support vector machines (Kabir et al. 2018)
- Artificial neural networks (Ashfaq et al. 2017)
- Ensemble of classifiers (Bukhtoyarov and Zhukov 2014)

Information Theory-based Methods is centred on the quantification of information and redundancy analysis. The two most popularly used information technology measurements are the Shannon Entropy (Bereziński et al. 2015; Behal and Kumar 2017) and the Kullback–Leibler distance (Xie et al. 2016; Li and Wang 2012).

Evolutionary Computation Methods is a set of algorithms inspired by natural evolution. Its goal is to be able to learn and adapt like biological organisms. Some examples of these algorithms include:

- Artificial Immune Systems (Shamshirband et al. 2014)
- Genetic algorithms (Hamamoto et al. 2018)
- Differential evolution (Elsayed et al. 2015)
- Particle swarm optimization (Bamakan et al. 2016)

Hybrid Methods are combinations of the different methods described above. Some of the work which uses hybrid methods are by Bostani and Sheikhan (2017) and De Assis et al. (2017).

2.4.2 Machine Learning models for Anomaly Detection

Much research on Machine Learning models has been done in the field of Anomaly Detection. We have chosen the following five models from different categories of methods for their compatibility with our data and their processing speed.

Clustering Based Local Outlier Factor He et al. (2003) presented a new definition for an outlier, a Cluster-based Local Outlier. An example to illustrate this idea is shown in Figure 2.1, C1 and C3 are observably outliers. The proposed Cluster-based Local Outlier Factor (CBLOF) is the measure for identifying the outlier's physical significance. The measure this, data must be partitioned into clusters with any clustering algorithm. Then identify the large and small clusters. If the record is in a small cluster, its CBLOF is the minimum distance to the closest large cluster. Otherwise, CBLOF is the distance from the record to the cluster it belongs to.

The time complexity of the CBLOF algorithm is $O(n)$

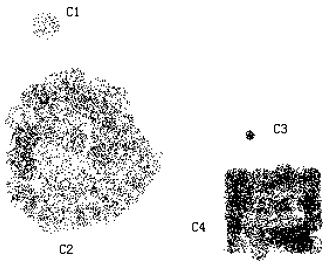


Figure 2.1: *Clustering illustration from He et al. (2003)*

Histogram-based Outlier Score Histograms are generally used to estimate the density of the distribution of univariate data. Goldstein and Dengel (2012) proposed a method of outlier detection using histograms. For data with more than one feature, a histogram is created for each feature. The type of histogram created could be a static bin-width histogram or a dynamic bin-width histogram. This depends on the distribution of the feature, data containing a large gap in the distribution will not be estimated well using a static bin-width histogram. Histograms for all features are normalised so that all features are on the same scale and the maximum height is 1.0.

The Histogram-based Outlier Score (HBOS) of an instance p in the dataset is calculated by:

$$HBOS(p) = \sum_{i=0}^{d} \log\left(\frac{1}{hist_i(p)}\right) \tag{2.1}$$

Where $hist_i(p)$ is the density estimation at p and d is the number of features in the dataset. The time complexity of the HBOS algorithm is $O(n)$ in the case of static bin-width histograms and $O(n \cdot \log n)$ in the case of dynamic bin-width histograms.

Isolation Forest Outlier Detector While most model-based anomaly detection approaches focus on creating a profile for normal behaviour to identify instances that do not fit the profile. Liu et al. (2008) proposes Isolation Forest (IForest), a model which explicitly isolates anomalies without profiling normal instances. Two parameters needed in building an isolation forest are the subsample size (ψ) and the number of trees (t). Isolation Forest is an ensemble of t Isolation Trees of ψ sample size. The Isolation Tree is built by randomly selecting features, and then partitioning samples into two groups at some value between the minimum and maximum of the feature. Samples that are of the normal behaviour will be common and therefore need more partitions in order to completely isolate it, giving it higher isolation depth. While samples that are anomalies or outliers will be isolated very early on, giving it lower isolation depth. Isolation forest has the time complexity of $O(t\psi \log \psi)$.

Figure 2.2 shows the process of isolating a normal sample X_1 and an anomaly X_0. X_1 needs twelve partitions to be isolated, while X_0 needs only four.

One-Class Support Vector Machine There are two approaches to One-Class Support Vector Machine (OCSVM), proposed by Schölkopf et al. (2001) and Tax and Duin (2004). Where

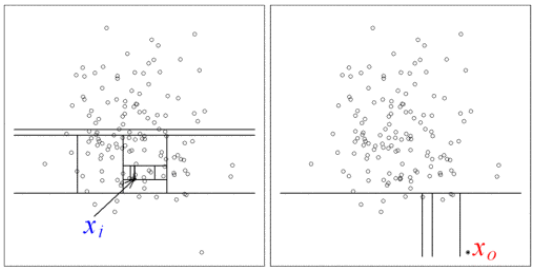


Figure 2.2: *Visualisation of sample isolation from Liu et al. (2008)*

normal SVM solves the problem of fitting a hyperplane with the maximum margin between different classes, OCSVM solves the problem of fitting a hypersurface which separates one class from all other classes. OCSVM are trained with datasets containing only normal instances.

The OCSVM proposed by Schölkopf et al. (2001) finds the hyperplane supporting the data with maximum distance from the origin. The outliers will lie below the hyperplane and closer to the origin. The OCSVM proposed by Tax and Duin (2004) finds the hypersphere supporting the data with minimum volume. Any points lying outside the hypersphere are outliers.

Principal Component Analysis Outlier Detector Principal Component Analysis (PCA) is a method for reducing data dimensionality while preserving as much information as possible. Principal Components (PCs) are new features which are a linear combination of the original features that capture the variance of the data. Data can have many PCs, the first PC captures the most variance in the data, the second PLC captures a lot of variance in the data but not as much as the first PC, and so on. The last few PCs represent the linear combination of the data with minimal variance. A model proposed by Shyu et al. (2003) detects outliers based on the assumption that observations which are outliers with respect to the first few PCs usually correspond to observations which are outliers on one or more of the original features. The observations with large values on the last few PCs will also reflect multivariate outliers that are not detectable based on original features alone. The authors claimed their approach can be used in real-time.

2.5 Related Work

Several researchers have analysed distinct types of attacks against ICS. Most research focuses on the communication between components, or communication in specific protocols. Some researchers have also suggested attack detection mechanisms for ICS attacks. On the other hand, researchers have studied commercial operating system vulnerabilities and its trend. Little research focuses on the vulnerability of commercial operating systems and their impact on ICS. This section will describe related work on ICS network vulnerabilities, Modbus vulnerabilities, ICS intrusion detection, and commercial operating systems vulnerabilities.

On PLC network security (Ghaleb et al. 2018), presents a security analysis of the communication between Siemens PLC and an engineering workstation. The author performed three diverse types of attacks: Replay attack, Man in the Middle attack, and stealthy command modification attack. These attack scenarios assume that the attacker is connected to the PLC and workstation via an Ethernet switch. All attacks were successful after using standard tools and techniques. This demonstrates the vulnerability of Siemens ecosystems. The attacker did not need to have knowledge of the process to successfully perform the three types of attack on the system. These scenarios usually occur after the attacker gained access to the network, a step which is not in the scope of their work.

Tzokatziou et al. (2015) exploited the communication between PLC and workstation on a different brand of PLC, ABB PM564 PLC. Their attacks include: initial access, reconnaissance, and exploitation. They had used a Teensy HID device for the initial access to the workstation. This is more realistic since most ICS are behind a firewall and a common point of access is via a removable device. The author analyses the packet characteristics and derived the byte signature of a STOP command. They successfully created a connection to the PLC and made the PLC accept the STOP. This is because the specific brand of PLC does not require authentication. The workstation also did not perform checks on the Teensy device acting as a keyboard. This is valuable as it shows the vulnerability in the workstation and the PLC operating systems.

Modbus is a commonly used protocol in ICS, its vulnerabilities have been demonstrated and categorised by researchers. An example of Modbus exploitation is in research by Alsabbagh et al. In this paper, the authors performed a stealthy false command injection attack on Modbus. They performed analysis on the Modbus packet and found the content to be easily understandable. Because Modbus has a predefined function code and no complex structure of its data frame, it is easy for an attacker to forge and inject a packet with a correct format. Different types of Modbus attacks have been grouped to form 17 attack categories in the work by Morris and Gao (2013). They gave each category's definition and generalised the result of the attacks.

To prevent being attacked, ICS must have the ability to detect an ongoing attack. This can be achieved by the deployment of intrusion detection systems on the ICS network. Two main approaches towards IDS are signature detection and anomaly detection. In their paper, Keliris et al. (2016) uses a support vector machine-based algorithm to detect abnormalities in their Hardware-in-the-Loop testbed. They were using one SVM to detect an ongoing attack and another SVM to categorise the attack. SVMs are trained using time signal patterns of the processes during normal operation. The result of the experiment shows that the SVMs were able to detect and categorised the attack. Although the author covers only the data from the process in their investigation, the same approach can also be used with network and host event logs.

Operating systems vulnerabilities often play a part in the exploitation of ICS. There are many studies on the vulnerability of operating systems popularly used in ICS such as Windows. Movahedi et al. (2019) has shown that Windows is most vulnerable to code execution and privilege escalation. Gu et al. (2022) showed 29 common vulnerabilities and exposures (CVE) of COM object data race that could be exploited for privilege escalation. Tigner et al. (2021) prove remote code execution can be achieved using reverse TCP trojan.

3 Design & Methodology

One of the challenges in conducting ICS research is securing an experimental ICS environment. It is considered unsafe to conduct experiments on an operating ICS due to the probability of catastrophic risks (Dehlaghi Ghadima et al. 2022). Researchers often use small-scale physical testbeds, virtual testbeds, or hybrid testbeds.

A small-scale physical testbed has physical equipment such as PLC, HMI, sensors and actuators used in the process. Examples of small-scale physical testbeds are the SWaT (Mathur and Tippenhauer 2016), the FACIES (Miciolino et al. 2015) and the WADI (Ahmed et al. 2017). A hybrid testbed still includes physical components such as PLC and HMI, but other components such as sensors and actuators are virtually simulated. Examples of the hybrid testbeds are the ones by Keliris et al. (2016), Rosa et al. (2017) and the VTET (Xie et al. 2018). In contrast, a virtual testbed simulates every component of the system. Examples of virtual testbeds will be discussed below. Different types of ICS testbeds are compared according to their fidelity, cost, repeatability and scalability by Shi et al. (2022), shown in Table 3.1. A virtual ICS testbed is used in this research for experimenting and collecting data on ICS attacks.

Table 3.1: *Testbed comparison (Shi et al. 2022)*

Testbed Type	Physical	Hybrid	Virtual
Fidelity	High	Medium-High	Low-Medium
Cost	High	Medium-High	Low
Repeatability	Medium	Medium	High
Scalability	Low	Medium-High	High

3.1 Existing Work

Morris et al. (2015) developed a virtual ICS environment for data logging. This testbed also implements levels 0 to level 2 of the Purdue model (Table 2.1). The process is simulated using Matlab/Simulink. The PLC is simulated using a Python script that mimicked the behaviour of the PLC ladder logic program. The testbed has two HMIs, one is a proprietary software GE iFix HMI and another is a Python tkinter program. The author performed Modbus attacks on the virtual testbed and compared its behaviour to a lab-scale ICS running the same physical process. They concluded that the virtual testbed behaves similarly to their physical ICS and could be used to model other types of ICS.

The ICSIM (Dehlaghi Ghadima et al. 2022) is a proposed simulation framework based on Purdue Enterprise Reference Architecture (Table 2.1). The current version of the framework covers the model from level 0 to level 3. ICSIM offers two environments for running the simulation: Docker for containers, and Graphical Network Simulator (GNS3) for containers and physical devices. ICSIM has its own code for HMI, PLC and the process. The physical process is simulated using python script, but the author suggested third-party software such as Matlab/Simulink can be used as well. The author successfully performed DDoS and MitM attacks on the ICSIM and was able to collect network data to analyse PLC request response time.

To exploit DoS vulnerability in Modbus protocol, Rahman et al. (2022) designed a virtual testbed which uses Modbus protocol for communication. The testbed implements levels 0 to level 2 of the Purdue model. The software FactoryIO is used to simulate level 0 of the model. OpenPLC is used to simulate the PLC in level 1 and ScadaBR simulates the HMI in level 2. The authors were able to perform DoS attacks on the testbed and produce results in traffic capture and time series plot.

The GRFICS framework (Formby et al. 2018) motivation is to make ICS security accessible for students. It implements levels 0 to 2 of the Purdue model. The researchers implemented a 3D visualisation for level 0 of the model using the Unity 3D engine. OpenPLC is used to simulate a PLC and an open-source software AdvancedHMI is used to simulate the HMI. The authors showed that the framework is capable of simulating ICS network attacks and ICS device attacks.

3.2 Requirement Identification

The purpose of our testbed is to show how ICS can be attacked using COTS and network vulnerabilities and to analyse the impact of the attack. Existing frameworks have demonstrated that this aim is possible by simulating ICS levels 0 to 2 (Section 3.1).

The user in the virtual testbed is a subset of real ICS users since we will only implement the core components of the Purdue model levels 0 to 2. An additional user in the virtual testbed is the researcher who will be attacking, collecting data and evaluating the testbed. Therefore the users of the virtual testbed are the HMI operator, the engineer on the workstation and the researcher.

3.2.1 User Stories

- As an HMI operator, I want to track the process parameters, so that I can analyse the state of the process.
- As an HMI operator, I want to issue commands to start and stop the process, so that I can ensure the process is working at an appropriate time.
- As an HMI operator, I want to adjust the control set point on the PLC, so that I can control the quality of the output and ensure safe operation.
- As an engineer, I want to develop the PLC ladder logic program, so that I can create the right solution for process control.
- As an engineer, I want to upload the ladder logic program on the PLC, so that I can keep the process control up to date.
- As an engineer, I want to have a connection to the management in the corporate network, so that I can send system status reports to management.
- As a researcher, I want to have exploit kits available to me, so that I can reduce time and effort in my attack development.
- As a researcher, I want to exploit COTS OS devices, so that I can demonstrate the impact of common vulnerabilities exploitation on ICS.
- As a researcher, I want to launch attacks over the network, so that my attacks are visible for evaluation.
- As a researcher, I want to collect the network traffic, so that I can perform an evaluation of the attack and use the data to create detection strategies.
- As a researcher, I want to collect the PLC log, so that I can perform an evaluation of the attack.

3.2.2 Functional Requirements

The functional and non-functional requirements for the virtual testbed are derived from the user stories.

R1 The HMI operators must have a dashboard containing comprehensive data from the PLC and process.

R2 The HMI operators must have a control panel for sending commands and modifying the control setpoint of the PLC.

R3 The engineer must have an integrated development environment (IDE) for developing a ladder logic program.

R4 The engineer must have a portal for uploading the ladder logic program on the PLC.

R5 The engineer must have a connection to the ICS network and the corporate network.

R6 The researcher must have a computer with the ability to download and run exploit kits.

R7 The researcher must be connected to the corporate network.

3.2.3 Non-Functional Requirements

R8 The virtual testbed must include devices with COTS OS.

R9 The virtual testbed must have logging mechanisms for both PLC and network packets.

R10 The ICS network must be a wired LAN.

R11 The corporate network should be wired or wireless LAN or WAN.

R12 The virtual testbed should be reproducible.

R13 The virtual testbed should be lightweight.

3.3 Tools

OpenPLC is an IEC61131–3 compliant open-source PLC for cyber security research. It is essential to use a PLC simulator that produces valid behaviours compared to real PLCs. OpenPLC has been evaluated against real PLC by Alves et al. (2014), and the result confirmed that the OpenPLC runtime behaves the same way as a Siemens S7-200 PLC working on the same process. Alves and Morris (2018) evaluated OpenPLC performance in scan time, Ladder Logic execution, SCADA Connectivity, and Modbus injection attack, compared to four commercial PLCs: Schneider M221, Siemens S7-1214C, Omron CP1L-L20DR-D, and Allen-Bradley (A-B) MicroLogix 1400. The authors concluded that OpenPLC behaves similarly to real PLCs in all four tests.

OpenPLC provides its own editor, the OpenPLC editor. This is used to program the PLC using Ladder Logic, satisfying **R3**. The ladder logic program can also be uploaded to the PLC via OpenPLC Runtime Webserver, satisfying **R4**

ScadaBR is an open-source HMI software. Both OpenPLC and ScadaBR use Modbus TCP protocol for communication, which makes both tools compatible. An important feature of the tool is its logging feature. ScadaBR polls and logs the values from the PLC continuously. The user can create a watchlist and graphical view of the PLC and process parameters, this feature satisfies **R1**. ScadaBR can be configured to produce reports of the logged PLC parameters, which satisfies **R9**. ScadaBR is used in ICS research for its aforementioned functionalities (Almas et al. 2014; da Silva and Cunha 2017; Silva et al. 2015; Rahman et al. 2022).

Wireshark is an open-source tool for packet capture and analysis. It can distinguish packets from different protocols, parse packet data into comprehensive fields, and map those fields to the packet's raw bytes. Wireshark can capture packets faster with more details compared to other network sniffing tools (Goyal and Goyal 2017).

Simulink is a MATLAB-based environment for modelling and simulating dynamic systems. Simulink is widely used in research on physical processes. The physical process chosen for this testbed is a simple water level control system. Many researchers have used Simulink to simulate hydraulic systems with reliable results (Rahman et al. 2022; Getu 2019; Song et al. 2017; Yigit and Selamet 2016). Therefore, Simulink is used to simulate the physical process in this virtual ICS.

Kali Linux is a Debian-based distribution for digital forensics and penetration testing. It contains around 600 tools and framework for vulnerability scanning and exploitation. This can help reduce time in attack development and satisfy **R6**.

VirtualBox is a cross-platform visualisation software. It allows us to run different COTS OS on our MacOS host in this research.

Docker is a platform that uses OS-level visualisation to allow users to package and deploy software in a lightweight container. We use Docker and VirtualBox together to produce a virtual testbed that is reproducible (**R12**) and lightweight (**R13**).

3.4 Design

The virtual testbed (figure A.1) is designed to cover levels 0 to level 4 of the Purdue model.

Level 0 of the virtual testbed is a process implemented in Simulink.

Level 1 of the model is a PLC simulated using OpenPLC runtime software.

Level 2-3 of the model is an HMI and an engineering workstation. Windows 10 is chosen as the COTS OS for the machines because all the software needed to implement HMI and the workstation is compatible with Windows OS. Windows also has many known vulnerabilities often exploited in ICS attacks (Falliere et al. 2011; Farwell and Rohozinski 2011; Cherepanov and Lipovsky 2016). Using Windows 10 as COTS OS for the HMI and the workstation satisfies **R8**. The HMI is implemented using ScadaBR and the workstation has OpenPLC Editor installed.

Level 4 of the model is the corporate network. This layer consists of machines used in different departments, e.g. sales, human resources, etc. The attacker is on Kali Linux and is assumed to have compromised a machine at this level. This satisfies **R7**.

3.5 Methodology

3.5.1 The Physical Process

The model used in the virtual testbed is a simpler version of the water tank model by Getu (2019). In their model, the tank has an inlet pipe and an outlet pipe. A PID controller is used to regulate the voltage of the pump that puts water in the tank. The water flows out of the tank as a function of height. The model is adapted by replacing the original control with the PLC (figure 3.1). Where the control adjusts the status of the inlet valve and outlet valve to keep the height of the water between the minimum and maximum levels. The height of the water in the tank at the time can be obtained from the equation:

$$h(t) = \int \frac{Q_{in} - Q_{out}}{A} dt \tag{3.1}$$

where Q = rate of flow (cubic meter/second) and

$$A = \pi r^2 \tag{3.2}$$

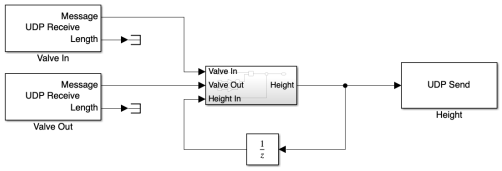


Figure 3.1: *The overall Simulink model*

Simulink periodically sends a UDP packet containing the height to OpenPLC and polls the value of each valve to assign flow rates. In this simulation, we set the sampling time of all Simulink UDP receivers and senders to 1 second to synchronise the communication between Simulink and OpenPLC.

3.5.2 The Programmable Logic Controller

OpenPLC has its own addressing scheme. The PLC address configuration in this testbed can be seen in figure 3.2. The addresses in OpenPLC have a format of '%location.type.byte.bit'. The location specifies if the variable will be in the input, output, or in memory. The types of variables in OpenPLC are a bit, byte, word, double and long. The last two values in the address specify the offset of the variable location. This is relevant when the PLC needs to communicate with Simulink and ScadaBR. In Modbus communication with ScadaBR (the HMI), the address field

#	Name	Class	Type	Location
1	Valve_In	Local	BOOL	%QX0.0
2	Valve_Out	Local	BOOL	%QX0.1
3	Start	Local	BOOL	%QX0.2
4	Stop	Local	BOOL	%QX0.3
5	High	Local	BOOL	%IX0.0
6	Low	Local	BOOL	%IX0.1
7	Master	Local	BOOL	%IX0.2
8	Height	Local	UINT	%IW0
9	High_Limit	Local	UINT	%QW1
10	Low_Limit	Local	UINT	%QW2

Figure 3.2: *The addressing of the PLC*

and location can be specified during ScadaBR data source set-up. This reduces the challenge in the variable address configuration during PLC programming. However, in the UDP interface used to communicate with Simulink, the address of the variable being polled is fixed to be sequential, starting from the first of each type. Only variables in output location (%Qx.x.x) can be polled. This resulted in many iterations of re-configuration for the addressing of PLC variables.

Ladder logic programming language is used to implement the PLC program (figure B.1). The "rails" in ladder logic represent the start and the end of each rung and a "rung" in ladder logic represents a rule. The rungs in the rails are executed sequentially. The ladder logic program implemented for the PLC in this testbed consists of 5 rungs.

- The first rung checks the variable Start and Stop to determine the PLC state. The rest of the rungs will use the variable resulting from this rung as the first "if" clause.
- The second rung checks whether the height of the water has reached the limit. It outputs results in the High variable (a Boolean).
- The third rung checks whether the height of the water has reached the limit. It outputs results in the Low variable (a Boolean).

- The fourth rung checks the High and Low variables to determine the state of the outlet valve.
- The fifth rung checks the High, Low and outlet valve variables to determine the state of the inlet valve.

The PLC runs on a Docker container with an OpenPLC runtime image. Once the container is up; the PLC can be reached on the OpenPLC webserver via port mapping from the Docker container to localhost (figure B.2). The OpenPLC server is where the program can be uploaded. The server also allow setting the polling interval, which is set to 1 second to synchronise with the Simulink model. OpenPLC has some support for Simulink, there is an option available for setting Simulink as the PLC "hardware". However, OpenPLC does not provide full integration support for Simulink, and the interface used to communicate with Simulink must be developed separately.

3.5.3 The Human-Machine Interface

ScadaBR is used as the HMI for this testbed. The OpenPLC runtime can communicate with ScadaBR using the Modbus protocol. To do this, we must first register the PLC as a "data source". In this scenario, the PLC acts as a Modbus master and ScadaBR is a Modbus slave. This is similar to the client-server communication model in HTTP. After setting up the data source, we can register each variable in the PLC as a "data point". During this step, we must inform ScadaBR of the location of the variable in the PLC address space (figure 3.2). The data points can then be inserted into the "watchlist" (figureB.3), where ScadaBR will keep polling data points in the watchlist every second.

ScadaBR also gives the option to create a graphical view of the data points in the watchlist. This helps the user of the HMI visualise the data points better. In this testbed, we created a graphical view to keep track of important PLC variables and the process (figure B.4).

ScadaBR provides the option to modify PLC parameters. However, the usability of this feature is quite poor. We decided to develop a small python application to simulate a control panel (figure B.5) and satisfy requirement **R2**.

3.5.4 The communication

The communication between OpenPLC and Simulink is one of the challenges faced in implementing the testbed. OpenPLC has full support for Modbus communication, but Simulink does not. However, Simulink has a model block with UDP send/receive functionality. The creator of OpenPLC has developed an open-source guidance code for host-based UDP communication with the PLC (Alves). We then reconfigured this code to work with a PLC and a Simulink model that is running on separate Docker containers. Injecting the script in either of the containers makes the communication overly complicated since both containers use their own image. In the end, the script for PLC-Simulink communication was made a separate interface running in its own Docker container. The three Docker containers (OpenPLC, Simulink, and the interface) are then put in the Docker bridged network. The interface acts as a middleman, polling data from Simulink and delivering it to the PLC and vice versa.

Because of the Docker container's inability to visualise Windows 10 OS, the workstation, HMI and the attacker machine were implemented using VirtualBox virtual machines instead. This gave us the advantage of functionality since using Kali Linux virtual machine as the attacker gave us more tools to attack the testbed, and ScadaBR does not support Docker containers.

Having parts of the testbed on Docker containers and other parts in separate virtual machines created difficulties in connecting the testbed. The Docker containers run in their own bridged network, separated from the host. Each virtual machine also needs some configuration on

VirtualBox to communicate with each other. The first part of the solution to this problem is to create a port mapping in the Docker containers where packets can be forwarded to the container from localhost. The second part of the solution is more complicated. Initially, the plan is to configure all virtual machines to share the host's network interface. Unfortunately, MacOS (the host OS) no longer supports this VirtualBox functionality. For this reason, two PfSense virtual routers are integrated into the testbed. The PfSense router is connected to the internet using the NAT networking mode by VirtualBox. This networking mode allows the virtual machine to communicate with the host, but the virtual machine is unreachable from the host and other virtual machines. The Pfsense routers are then connected to two separate host-only networks, which work similarly to a wired LAN, and allow communication between machines in the network. The workstation and HMI virtual machines are then put in the host-only ICS network. The attacker machine and the workstation are put into the host-only corporate network. This network setup satisfies **R5**, **R10**, and **R11**.

The host-only network mode creates a new network interface on the host that enables virtual machines to communicate with the host, other virtual machines, and the internet using PfSense virtual router as the gateway. PfSense uses a DHCP server to allocate IP addresses to its client dynamically. We assigned 192.168.88.0/24 to the ICS network interface and 192.168.89.0/24 to the corporate network interface. The host will automatically be given the lowest address available, other machines will be assigned IP addresses dynamically. This becomes a problem because some programs running on the testbed use fixed IP addresses as input. Thus, the workstation, the HMI and the attacker machine are given static DHCP leases. The testbed connectivity can be visualised as in Figure 3.3.

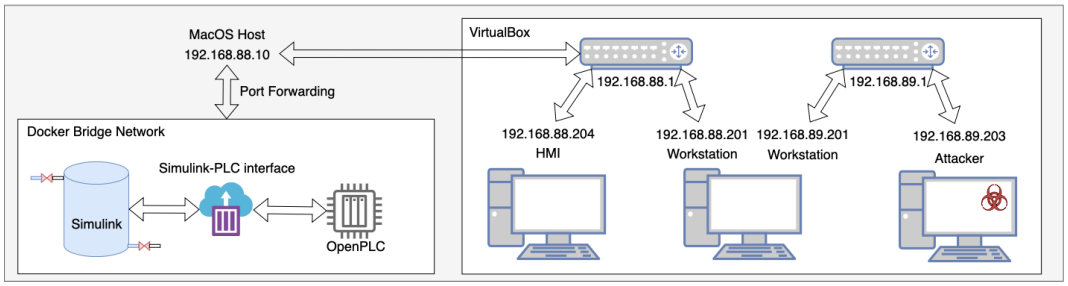


Figure 3.3: *The testbed network*

3.5.5 Data Logging

To satisfy requirement **R9**, we configured ScadaBR to generate an hourly report of all PLC variables in the watchlist. The report is saved in the application in the form of Comma Separated Values.

As mentioned in the previous subsection, the host-only network mode creates new interface on the host machine. This interface received equal traffic as the router. To capture the network traffic on both the ICS network and the corporate network, we use WireShark on the host machine interface. The traffic captures are saved on the host in the form of PCAP files.

4 Implementation

4.1 Threat Model

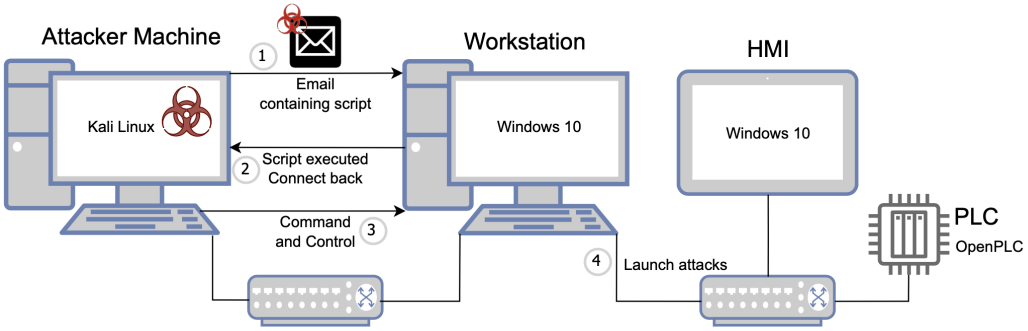


Figure 4.1: Threat model

Figure A.1 presents the hierarchical representation of the testbed. Where the attacker sits in the corporate network on level 4, the targeted workstation in levels 2-3, and the targeted PLC on level 1. The aim of the attacker is to gather information on the ICS, compromise the physical process, and disrupt the availability of the ICS.

Figure 4.1 presents the simplified steps of the attack. This model assumed social engineering attack has been done successfully. The attacker is already in the corporate network, and they have acquired the emails of the site employee. The attack can be divided into two parts. First, the attacker compromises the workstation and gains access to the ICS network through the connection between the engineering workstation and the corporate network. Then, acting through the workstation, the attacker performs several attacks on the PLC.

The workstation's firewall and anti-virus have **not** been modified prior to the attack. The attack assumes at least one engineer on the workstation will click the link attached to the internal spear-phishing email sent by the attacker.

4.2 MITRE ATT&CK Framework

The MITRE ATT&CK framework is the curated knowledge base and model for cyber adversary behaviour, detailing the adversary attack's life cycle. This framework provides a common taxonomy and categorisation of the adversary actions (Trellix). The framework is used across multiple disciplines, including intrusion detection, threat hunting, security engineering, threat intelligence, red teaming, and risk management (Strom et al. 2020).

The attack in this experiment falls into two ATT&CK framework iterations. The attack on the workstation through the corporate network follows the ATT&CK for Enterprise framework, specifically the ATT&CK for Windows. This framework focuses on the attacker's behaviour

in the Windows environment. The attack on the PLC through a compromised workstation follows the ATT&CK for ICS framework which focuses on the attacker's behaviour within the ICS network.

The ATT&CK framework for Windows and the ATT&CK framework for ICS have different tactics and techniques. However, there are still some overlapping tactics. Table C.1 maps the tactics and techniques used in this experiment to both MITRE ATT&CK framework. The two following sections will discuss the attack in more detail with respect to the order of execution and the ATT&CK frameworks.

4.3 Compromising the Workstation: MITRE ATT&CK for Windows

This section explains each step the attacker took with the goal to gain privilege and control of the Windows workstation.

4.3.1 Resource Development

During the execution of the resource development tactic, the attacker developed (ATT&CK T1587) and acquired (ATT&CK T1588) the following resources.

The payload stager. The attacker developed the script called "update" to act as a stager for the MSFvenom payload. The stager executes as a command line script which calls Windows PowerShell to execute the encoded command. The encoded command has two important functionalities: Windows defender evasion and MSFvenom download. Because Windows Defender can detect the signature of MSFvenom Meterpreter payload. This makes downloading the payload impossible without tampering with the Windows Defender.

The stager has two options for evading the defender. If the user has administrator privileges, the stager can simply execute a PowerShell command to exclude the folder from malware scans. If the user has no administrator privileges, the attacker will proceed following the approach by Ortiz (2020b). The stager calls Windows API to load "amsi.dll", the Windows Antimalware Scan Interface library. The load function will point to the relative address of the library. The stager then calls Windows API to get the address of the "Amsiscanbuffer" procedure, residing in AMSI library. Once the address of the scan procedure is known, the stager patches the procedure's return value with "AMSI_RESULT_CLEAN", which corresponds to no malware being detected.

The stager itself also evades defences using the file obfuscation technique (ATT&CK T1027.004). The stager command is encoded and will be decoded by PowerShell before compilation. The signature of the payload stager might still be marked as suspicious because it contains commands such as VirtualAlloc, CreateThread, etc. For this reason, the stager's command is obfuscated before encoding using string obfuscation techniques (Danielbohannon) to deviate the command from the common signatures.

After successfully modifying the Windows Defender malware scan, the stager downloaded the MSFvenom payload from the Apache server hosting on the attacker's machine and passed the execution to it.

The MSFvenom payload The attacker acquired the Metasploit framework as one of the main tools that will be used in the attack. Metasploit framework contains modules for vulnerability exploitation, payload generation and auxiliary functions. MSFvenom is the payload generation module from Metasploit. Using MSFvenom, the attacker generates a reverse TCP stager for a Meterpreter payload. Meterpreter is a part of Metasploit, it is used to create a connection between the victim and the Metasploit MSFconsole running on the attacker side. Meterpreter embeds

itself in the executable and allows more exploitation to be done at a later stage using Metasploit modules.

Metasploit uses its signature reflective DLL injection technique (ATT&CK T1620) to evade defences. This is a technique developed by Stephenfewer. The general overview of this technique is to have an actor process download the malicious DLL into memory, copy the DLL onto the target process, and call its self-loader function, which will load its binary into the target memory to be executed.

The ICS attack scripts The attacker developed three scripts for the attack within the ICS network. These are simpler than the first two payloads due to there being less defence in the ICS device's operating system. The functionality of each script will be discussed in their respective subsections in the next section.

The ICS scripts will be downloaded from the attacker's Apache server after the attacker gained a Meterpreter connection to the workstation with at least Administrator level privilege at a later step. The scripts will be downloaded onto a well-hidden folder on the workstation. Defence evasion will not be needed because the content of the scripts is python codes that are not suspicious to the Windows operating system.

4.3.2 spear-phishing and payload execution

This step is based on the assumption that the attacker has an email list of the engineers working on the workstation. The attacker sent the phishing email to the victims (ATT&CK T1534), as shown in 4.2. When the victim clicks on the link in the emails, the payload stager will be downloaded from the attacker's Apache server. The payload stager waits for the user to click on the downloaded file (ATT&CK T1204.002), or for the browser to automatically open it.

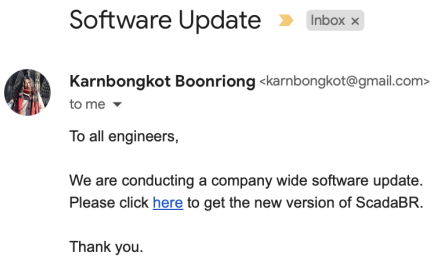


Figure 4.2: *Example phishing email*

The payload stager executes as a shell script (ATT&CK T1059.003) which calls onto PowerShell script (ATT&CK T1059.001). While the stager is being executed, the process windows will be hidden from the user. Therefore, the user is not aware of the process being executed on the workstation. This is the hiding window technique (ATT&CK T1564.003) for defence evasion. No other action from the user is required.

After the payload stager and the MSFvenom payload finished their execution, the attacker should have a Meterpreter connection with the workstation. From this stage onward the attacker has established command and control (ATT&CK T1095) with low privilege on the workstation.

4.3.3 Privilege Escalation

The attacker has already established a foothold in the systems from the previous step. However, to complete the attack life cycle, the attacker will require a higher level of privilege in order to control more components of the victim's systems. The attacker exploited Windows local privilege

escalation vulnerabilities to gain ideally SYSTEM-level privilege, or at least Administrator level privilege. Three disclosed zero-day vulnerabilities were chosen as exploitation targets for this step. The attacker will attempt to exploit each of the vulnerabilities until the attacker's privilege is escalated (ATT&CK T1068). This section will first provide background information on the Window drivers the vulnerabilities exist within, then provide more insight into each vulnerability.

Win32k.sys is a Windows kernel-mode driver, it is responsible for the graphical device interface and window management. The Win32k driver deals with graphical rendering, window objects, cursor objects, and other such objects and functionality. This makes Win32k one of the most essential parts of Windows systems. This is one factor for Win32k to be the ideal target, it will always be present and running on any Windows machine with a graphical interface.

The Windows processor can run in two modes: user mode and kernel mode. Applications run in user mode, and core system components run in kernel mode. The Win32k is a kernel mode driver, which implies that it will run in kernel mode with the highest privilege (SYSTEM). A user thread can call the core system function using Windows API. This will cause the processor to switch from user to kernel mode, and the user thread will wait for the "kernel-mode callback" to return control to the user side. Control could be switched multiple times. This gives the attacker a window of opportunity to corrupt the process, and consequently, escalation of privilege.

- **CVE-2021-40449** or the Win32k NTGdiResetDC use-after-free vulnerability. This vulnerability lies in a kernel-mode function which resets a specific data structure in one of the Win32k objects by releasing the old object and creating a new one. In calling this function from the user side, the user must pass the handle to the object as the parameter. The processor switches to kernel mode and performs its operations. During the execution, one of the kernel functions will make a kernel-mode callback, and the kernel side waits for the user action to finish its execution. The attacker can use this opportunity to make a second call to the same function using the same handle, while the original call is still in progress. The attacker lets the second call finish its execution; this means the handle will be freed. The attacker populates the memory referenced by the handle with an attacker-controlled value and finishes the execution of the original function call. The original function call still uses the same handle without validation, but that handle is now referencing attacker-controlled value. This gives the attacker the ability to manipulate the state of the kernel and gain kernel privileges (Raiu 2021; PacketStormSecurity a).
- **CVE-2022-21882** or the Win32k ConsoleControl Offset Confusion. This exploit is much more complicated than CVE-2021-40449. In short, Windows window objects can be created with extra data. There is a flaw wherein the extra data field of a window object can be manually modified to contain an attacker-controlled value. Then, during a kernel mode call to convert a window object to a console window, the attacker-controlled value will be treated as offset to a kernel mode address. This can be leveraged to achieve an out-of-bounds write operation, eventually leading to privilege escalation (Stefan Nicula 2022; PacketStormSecurity b).

The Windows Print Spooler is another one of the core system components in a Windows system. It manages the printing process. The print spooler is present in every Windows machine, and it will run continually from system startup to shutdown. When the user calls the interface to print a document, the print job is spooled to a predefined directory, "spool directory." This directory can be unique for the different print providers (printer), the spool directory is saved in the Windows registry. By default, a user without Administrator privileges can create a new local printer and configure its spool directory. The spooler can also be configured to load a specific library for the printer.

- **CVE-2022-21999** or the SpoolFool Privilege Escalation utilises the functionality mentioned above. The attacker will create a new local printer and then assign the spooler directory to be inside the printer driver directory, inside the system directory. So that later they can insert malicious DLLs into the directory. This is because the spooler will only load a

library if it is in the trusted territory, the system directory. During directory assignment, the spooler will validate that the spool directory path does not match the path to the system directory. The attacker can bypass this validation by using the directory junction and UNC path for the spool directory. The attacker can then write the malicious DLLs to the spooler directory. At this point, the spooler will have to be restarted to create the new printer and load malicious DLL files. This can be done following the approach by Mata (2021). After the restart, the malicious DLLs will be loaded and running with SYSTEM privileges inherited from the spooler process (Lyak 2022; PacketStormSecurity c).

The attacker acquired the exploits for the vulnerabilities from the Metasploit framework. The exploits are written in a DLL, which will be downloaded onto the victim over the current Meterpreter connection (TCP-based), and then executed using the reflective DLL technique.

From this step onward the attacker has established a foothold with high privilege on the workstation and will be able to use Windows resources as if the attacker were the system itself.

4.3.4 Persistence

The attacker has successfully gained a foothold and privilege in the workstation. However, if the victim shutdown or restart the workstation, the process that the attacker has created would be killed. The attacker must perform a persistence technique to maintain their foothold within the workstation. The attacker performed both of the following techniques to ensure persistence and backup.

Persistence in the system startup registry (ATT&CK T1547.001) Registry startup keys or registry run keys are sets of keys that will be run automatically at system startup. The attacker moved the payload into a non-suspicious location in the system and then set up a startup key that points to it. Whenever the workstation is turned on, the payload will be executed and will keep trying to connect back to the attacker. This technique is in fact the most common form of persistence (cyborgsec3 2022). It does not require a complicated setup and only requires administrator privilege. Figure 4.3 shows the attacker's key in the run key registry.

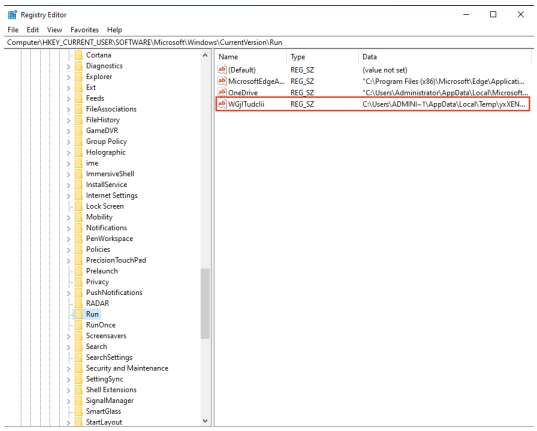


Figure 4.3: *The attacker's registry run key*

Persistence by creating a remote desktop user (ATT&CK T1136.001) The attacker has created a backup form of persistence in case the payload execution failed at start-up. A remote desktop allows the attacker to connect to workstations from afar. With the current Meterpreter connection, the attacker opens a reverse shell connection to the workstation. The attacker then uses normal shell commands to create a new user on the workstation. The attacker used discovering permission group (ATT&CK T1069) technique on the workstations to see if there is already a remote desktop user group. If there is, the newly created user is added to the group. If

not, the attacker used the shell command to modify the fDenyTSConnections key in the registry to enable remote desktop. It should be noted that this form of persistence is not as stealthy as registry keys. If the attacker tries to access the workstation through a remote desktop while the user is actively using the desktop, the user will receive a notification as shown in Figure 4.4.

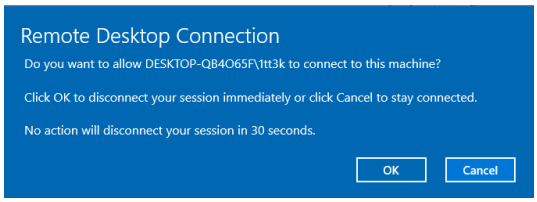


Figure 4.4: *The system's remote desktop connection notification*

4.3.5 Credential extraction from SAM dump

This extra step is done to ensure that the attacker collects any information they might need in the future. The attacker extracted the username and hashed passwords of the users from the workstation. The user's password is stored in a Security Account Manager (SAM) database file, mounted on the registry. SYSTEM privilege is needed to access the file. The passwords are hashed in NTLM format (Figure 4.5) which can be cracked using John the Ripper password cracker software.

```
1tt3k:1000:aad3b435b51404eeaad3b435b51404ee:c705696627d5de4c57e4e78e01a14eaa:::
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:d0a918bc8ea56b9213810b853f51b9b1:::
```

Figure 4.5: *The SAM password hashdump*

The last tactic that the attacker used on the workstation is discovery. The attacker performed network sniffing (ATT&CK T1040) to discover the devices in the ICS network using the workstation network interface. This tactic can be merged with the reconnaissance tactic for ICS. The life cycle of the attack on the workstation is completed.

4.4 Attacking ICS network: MITRE ATT&CK for ICS

4.4.1 Reconnaissance

The attacker now operates within the workstation, which is connected to the ICS network. Using the workstation's network interface, the attacker sniffed the ICS network and gain insight into the process. The attacker first used the Nmap tool to discover devices and open ports in the network (ATT&CK T1505). Nmap provides vulnerability scanner scripts which help the attacker identify vulnerabilities on the machines in the network.

Scapy (Secdev 2022) is a Python-based command line tool. It can be used to discover network devices, scan the network, sniff on network traffic, and forge and manipulate packets. The attacker used their Meterpreter connection to secretly install Scapy onto the workstation, then capture the ICS traffic to file. The attacker studied the traffic capture to learn the characteristics of normal traffic and discover windows of vulnerabilities. The attacker is able to profile devices based on the type of packet the device sends out. The vulnerabilities discovered during this step will be discussed in the following section, together with their exploitations.

4.4.2 Inhibit Response Function

ICSs are monitored in real-time. There are usually safeguards, alarm systems and fail-safe to ensure the least damage to the process in events of failure. When the attacker proceeds to the next tactic, impairing the process, the safety response mechanisms will be triggered. There are several techniques an attacker might employ to disrupt the ICS ability to respond to the unsafe state. These techniques, used together with techniques to impair the process, will give the attacker control of the ICS.

ARP Cache Poisoning (ATT&CK T0830) is a common attack in a local area network. Address Resolution Protocol (ARP) is a protocol used to map an Internet Protocol (IP) address to a Media Access Control (MAC) address. These mappings are stored in the ARP cache. ARP operates in the data link layer, layer 2 according to the OSI model, it does not provide any authentication mechanisms.

The steps to performing ARP cache poisoning in this ICS network involve:

1. Obtained the legitimate MAC addresses of the victims (Figure 4.6a).
2. Forge ARP packets using Scapy, replacing the legitimate MAC address with the attacker's MAC address.
3. Periodically sent out the forged packets to ensure that the victim's ARP cache stays poisoned (Figure 4.6b).
4. Return the network to the normal state by forging and sending out ARP packets with legitimate addresses (Figure 4.6c).

During the ARP cache poisoning attack, all packets in the ICS network will be sent to the attacker. The attacker holds a record of the correct MAC address for each IP address, so they could decide which packet to forward to the intended destination, and which to drop.

145…	921.195770	PcsCompu_88:85:88	a6:83:e7:d3:3f:64	ARP	60	Who has 192.168.88.10? Tell 192.168.88.201
145…	921.195781	a6:83:e7:d3:3f:64	PcsCompu_88:85:88	ARP	42	192.168.88.10 is at a6:83:e7:d3:3f:64
145…	921.196583	PcsCompu_88:85:88	PcsCompu_ff:ca:89	ARP	60	Who has 192.168.88.204? Tell 192.168.88.201
145…	921.201368	PcsCompu_ff:ca:89	PcsCompu_88:85:88	ARP	42	192.168.88.204 is at 08:00:27:ff:ca:89

(a) The attacker discovers MAC addresses of the victims

150…	940.089364	PcsCompu_88:85:88	PcsCompu_ff:ca:89	ARP	60	192.168.88.10 is at 08:00:27:88:85:88
150…	940.089685	PcsCompu_88:85:88	a6:83:e7:d3:3f:64	ARP	60	192.168.88.204 is at 08:00:27:88:85:88

(b) The attacker sends out forged ARP packets

685…	2745.196422	PcsCompu_88:85:88	PcsCompu_ff:ca:89	ARP	60	192.168.88.10 is at a6:83:e7:d3:3f:64
685…	2745.196649	PcsCompu_88:85:88	a6:83:e7:d3:3f:64	ARP	60	192.168.88.204 is at 08:00:27:ff:ca:89

(c) The attacker Re-ARP the victims

Figure 4.6: *ARP poisoning packets captured by Wireshark*

Read Bombing is a sub-technique for Denial-of-Service attack (ATT&CK T0814). This is a protocol-specific attack. During reconnaissance, the attacker discovered that the PLC uses Modbus TCP protocol to communicate with the HMI. Authentication is not required for a Modbus TCP connection. Therefore, the attacker made a connection to the PLC and flooded it with valid Modbus read requests. In the case that the ICS has its own security implemented to block an unauthenticated device from making a connection, the attacker will still be able to readbomb the PLC using the existing connection between the PLC and the workstation. The PLC will try to reply to all read requests causing it to overload.

Synflooding is another sub-technique for Denial-of-Service attack (ATT&CK T0814). Unlike readbombing attack, the SYNflooding attack is based on TCP protocol. The attacker knew the IP address of the PLC, and the common knowledge that the Modbus port is 502. The attacker launched a DDoS attack on PLC, flooding PLC with SYN packets and disrupting the communication between the PLC and its Modbus slave.

(a) Read Discrete Inputs request signature *(b) Read Holding Registers request signature*

Figure 4.7: *Packet signatures*

With the techniques mentioned in this subsection, the attacker could stop the PLC from responding to commands or requests in time. This takes away the control the engineers have over the PLC, and blocks any value updates the PLC sends to the engineers. It disrupts the availability of the ICS to respond to unsafe states and ensure the impact of the attack (ATT&CK T0826).

4.4.3 Impair Process Control

The last stage before physical impact is to impair the process. The attacker has observed the data being sent over the traffic and has gained more insight into the network characteristics. The relevant observations the attacker made are the followings:

- The attacker observed that the PLC and HMI communicate in loops. One loop consists of a read request for every type of Modbus data type: discrete inputs, coils, input registers, and holding registers. The loop gets repeated infinitely, and the requests and responses happen very quickly (around 1 millisecond). However, as shown in figure D.1, the attacker observes a one-second window between the last ACK of the loop and the first request of the new loop. The attacker could launch an attack in this window.
- The attacker observed the response of every read request to learn about the variables in the PLC. The attacker observed that the value of the PLC input register is constantly varying. The attacker also observes that the value of PLC input register does not go lower or higher than the values in the PLC holding registers. The attacker concluded that the values in the holding registers must be the control set point for the process.
- The attacker observed that the Modbus transaction identifier gets increased by one every single request. Having four requests per loop, the transaction identifier at the start of a loop is four unit more than the start of the previous loop. The attacker also knows that in a TCP communication, it is crucial for the packet to contain the correct sequence and acknowledgement number, which can be obtained from the ACK packet.
- The attacker observed that the byte sequence in the Modbus layer of a read request packet does not change over time (except the bytes representing transaction identifiers). The attacker can obtain the byte sequence and use it as a signature to identify a read request. The signature of the relevant read request is shown in Figure 4.7. This only applies to the specific PLC, because the bytes that formed the signature represent specific information such as variable offsets, unit identifier, and the number of the parameter being requested.

The attacker decided to make an unauthorised command (ATT&CK T0855) to modify the parameter (ATT&CK T0836) of the PLC. This will be done by forging and injecting packet with a write holding register command to alter the control set point in the holding register. As the PLC is controlling a physical process, changes to the control set point will likely affect the physical hardware. The attacker applied similar approaches to the Stuxnet worm, setting the control to an extremely high and extremely low value for a continuous block of time, aiming to damage the physical device (ATT&CK T0879).

With the observations mentioned above. The attacker developed a Python script with Scapy to perform the following algorithm.

1. Generate a random extreme value for writing to the PLC register.

2. Using Scapy, sniff on the traffic until a packet with the read discrete input request signature is captured. This signals the start of the loop.
3. Extract the byte after the end of the TCP layer definition but before the occurrence of the read signature. This byte is the Modbus transaction identifier. Increase it by four (base 10) to get the transaction identifier the PLC will be expecting at the start of the next loop.
4. Continue to sniff the traffic until a packet with the read holding register request signature is captured. This signals the last request-respond pair of the loop. The loop will end with an ACK from the HMI to the PLC to acknowledge the PLC response.
5. Continue sniffing until a packet is captured with TCK ACK flag = 1, IP source = HMI and IP destination = PLC. This is the last ACK of the loop.
6. Forge a write holding register request packet with the random value generated in step 1, the transaction identifier obtained from step 3, and the sequence-acknowledgement number acquired from step 5.
7. Inject the packet into the ICS network interface on the workstation.

The attacker repeatedly executed the algorithm to continually modify the control set point. However, the injection will not be successful in every single communication loop. This is because the attacker would need to forge a new packet every loop to update the values in each packet layer. This process takes at least 150 milliseconds. The attacker has around 1 second to forge the packet and inject it. If some of the steps in the algorithm are delayed and the attacker missed the 1-second window, the injected packet will be considered invalid by the PLC and will be discarded.

Figure D.2 shows the ICS traffic during the attack. The first frame circled in red is the injected packet that the PLC receive and responds to. The second circled frame is the legitimate read request sent from the HMI to the PLC that is now classified as a TCP retransmission (because of the duplicate sequence and acknowledgement number). The HMI will not get the correctly formatted response from the PLC, therefore it terminates the connection with the PLC, and restarts a new connection with the PLC after.

At this stage, the physical process is impaired and the safeguard is disrupted. The attacker gained persistent access to the workstation inside the ICS network with SYSTEM privilege. The impact is made and the attack life cycle is completed.

5 Evaluation

In this chapter, we will evaluate the effectiveness of the attack and the virtual testbed implementation. We will achieve this by analysing the result of the attack in two domains: the success and the effect on ICS.

The **the virtual testbed** is evaluated on its ability to exhibit behaviour change during the attacks. (**Aim 1**)

The effectiveness of **each attack** is determined by its impact on the network measurements and the state of the PLC. We will focus on the effect of the ICS because it is the main objective of the research. (**Aim2**)

The attack was implemented in two parts: compromising the workstation and attacking the ICS network. The success of **the full attack chain** is determined by the success of both parts. The success of each part is determined by its outcome, compared to the intended outcome:

- The attack on the workstation must gain the attacker's foothold, SYSTEM privilege, and persistence.
- The attack within the ICS network must block the HMI operator from sending commands to the PLC, alter the control setpoint, and show an impact on the physical process.

We will separate the rest of the evaluation into four parts: the trial setup, the evaluation of ICS attacks, network anomaly detection on ICS, and the evaluation of the Windows attack and its detection.

5.1 Experimental trial setup and data collection

During the experiments, the virtual testbed is left running without any involvements or modifications other than the attacks being performed. The experiments are done in the manner specified below.

The ICS baseline set is captured prior to the attack. We let the virtual testbed run its normal operation for two hours while collecting network traffic and PLC logs. This forms a dataset for the normal ICS behaviour which will be used to establish the baseline.

The ICS training set is the combination of six sets of traffic capture with a 0.05% contamination rate. There are six types of ICS attacks mentioned in chapter 4. For each type, we set up a one-hour trial where the attack is launched periodically for random intervals. The volume-based attacks are generally launched for shorter intervals because of the amount of packets they generate. The contamination rate is calculated from the number of seconds where there is an attack divided by the total time in seconds.

The ICS test sets consist of one full chain attack set and six individual attack sets, both are used to evaluate the anomaly detection models' performance. The full chain attack set is captured throughout the attack chain for one hour and twenty minutes. Each attack is executed in sequence with random breaks in between. The individual attack sets are used both in testing and plotting the result. We executed the attack once for an interval of five minutes maximum. We timed the attack so that the total normal traffic time in each set is around an hour. This will ensure the

contamination rate is appropriately low (approximately 0.05%), and the attacks will be visually pleasing in evaluation plots.

The corporate datasets consist of the baseline set, the training set, and three test sets. The attack is performed and captured in a similar manner to the ICS datasets.

5.2 Experimental Results

This section will evaluate the effect of each attack in the same sequence they were implemented. The success of the ICS attack will be discussed in the last step of the attack.

The network traffic will be measured using the following metrics:

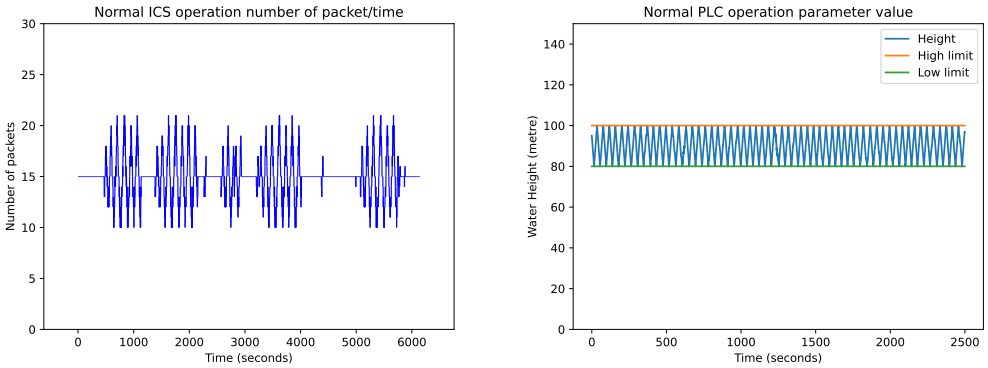
- Number of packets/Time
- Number of different destination ports/Time

The PLC state will be displayed in the format of:

- Water height value/Time
- Water height limit value/Time

5.2.1 Establishing Baseline

To effectively analyse the impact of the attack on the ICS network and the PLC, we first have to establish the behaviour of the network during normal operation.

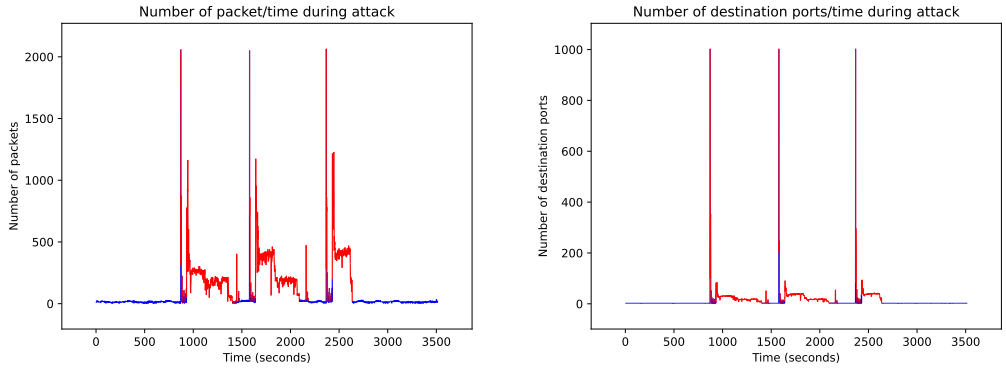


(a) Normal ICS operation number of packets over time *(b) Normal PLC operation parameter value*

Figure 5.1: *ICS baseline behaviour*

The network frequency Figure 5.1a shows the number of packets over time during normal ICS operation. Baseline traffic consists of the communication between the PLC and the HMI and occasional ARP requests between the two devices. The average number of packets over time is correlated with the number of packets in one PLC communication loop (16 packets). This is because during implementation we set the polling time between ScadaBR and the OpenPLC to be one second. However, the communication loop does not start and end according to the clock second, and sometimes there are delays. This resulted in the frequency change occurring at different points in time during normal operation.

The PLC parameters Figure 5.1b shows the value of the PLC parameters during its normal operation. The PLC has 10 parameters overall, however, we will only focus on the parameters relevant to the attack. We can observe that the water level will always be between the low and high limit parameter.



(a) Number of packets over time during reconnaissance *(b) Destination ports over time during reconnaissance*

Figure 5.2: *Impact of reconnaissance on the ICS network*

5.2.2 Reconnaissance

The attacker uses two techniques during the reconnaissance step of the attack: active scanning and passive sniffing. While passively sniffing on the network using Scapy does not generate more traffic in the network, active scanning using Nmap generates noticeably more traffic in the network.

Figure 5.2a shows the behaviour of the network during the active scan. Where the area that is highlighted in red indicates an ongoing attack. We can see a pattern that occurs for three iterations, each iteration of this pattern corresponds to a scan of each host on the ICS network: the router, the PLC, and the HMI.

During an active network scan, the Nmap tool will attempt to make a TCP connection on all ports in a host, this step maps to the first large spike at the beginning of the pattern. All the open ports on a host will acknowledge the connection, giving the second smaller spike in the pattern. As a result, the tool is able to determine which port is open. The area of increased traffic following the two spikes is mapped to Nmap vulnerability script execution.

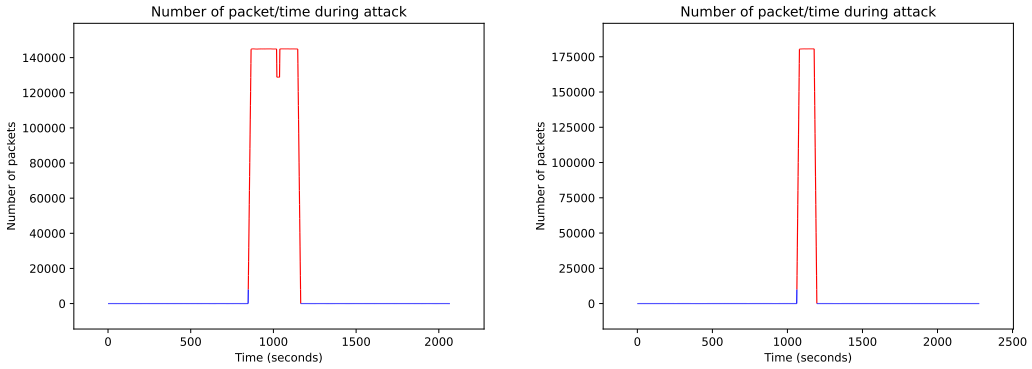
Another domain of the network that is sensitive to reconnaissance is the number of different destination ports at a time. During normal operation, there are only two main destination ports the packets are being delivered to, the PLC port 502 and a randomly assigned port on the HMI. During the active scan, the Nmap tool attempt to connect with all ports on all hosts. Therefore the number of different destination ports at the time will increase rapidly. This effect can be seen in figure 5.2b.

5.2.3 DoS attacks

DoS techniques were performed in the attack during inhibit response function step. The sub-techniques used in the experiment are PLC readbombing and SYN-flooding. Both of the sub-techniques attempt to achieve the same effect using a different protocol. Therefore, both attacks generate a similar effect on the network. This is visible in the domain of packets over time, as shown in Figure 5.3.

5.2.4 Injection attack

The injection attack was performed to alter the PLC parameter and impair the process. The impact of an injection attack lies in more domains than volume-based attacks. Figure 5.4 shows



(a) Impact of readbombing attack on the ICS network *(b) Impact of SYN-flooding attack on the ICS network*

Figure 5.3: *Impact of DoS attack on the PLC*

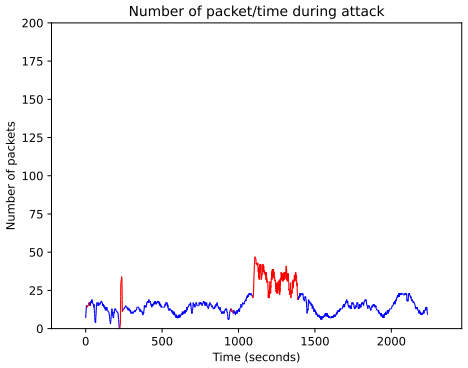
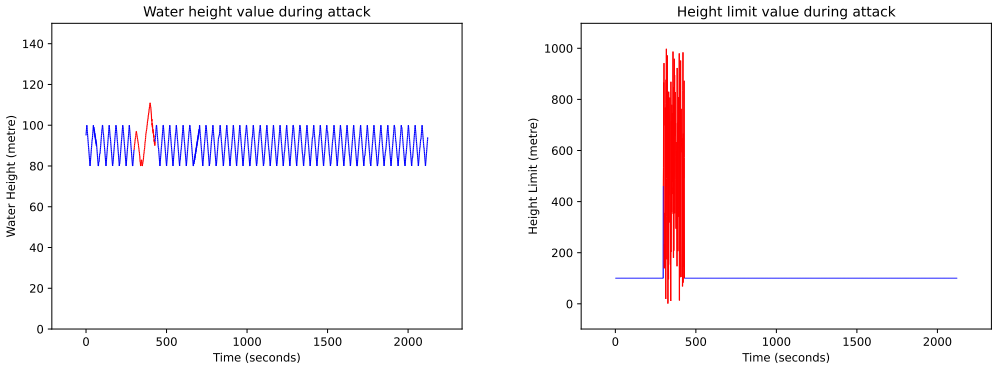


Figure 5.4: *Impact of injection attack on the ICS network*

the impact of injection attack on packets over time domain. Compared to the three previous attacks, the injection attack only slightly increased the number of packets over time. The first red spike at around T = 250 seconds comes from ARP poisoning. Despite only one packet being injected per communication loop, every injected packet is followed by multiple transmission attempts from the HMI, connection termination, and connection re-establishment. Hence, the number of packets over time is increased during the re-transmission and re-connection attempt.

The main effect of an injection attack is on the PLC parameters. The PLC height limit was the parameter we have concluded to be a control set point in Subsection 4.4.3. The height of the water in the process is dependent on the height limit parameter. In the attack, we rapidly alter this limit to extremely high and low values in every communication loop. We can observe the effect of the limit modification in Figure 5.5b.

However, the attack did not impair the process as severely as expected. Figure 5.5a shows the water level rising to an abnormally high level only once, despite the limit being altered multiple times. This shows a flaw in the attack. We as an attacker did not take into consideration the flow rate of the water in the tank. Altering the limit of the water in the tank will not impact the water level unless the water level is close to the limit. In their future attack, when generating a new limit for the injection, the attacker should take into account the water level in the communication loop. This can be done by adding a step to sniff for a packet with the "Read Input Register" function signature and extracting the water height parameter from the PLC response before



(a) Water height value during injection attack *(b) Height limit parameter value during injection attack*

Figure 5.5: *Impact of injection attack on the PLC*

generating a suitable limit value.

5.2.5 Evaluating the success of the ICS attack

As mentioned in Chapter 4, the ICS operator will be able to notice the water rising to an abnormal height during an injection attack. Therefore the injection attack must be performed along with a denial of service attack to prevent the ICS operator from becoming aware of the ongoing attack.

While performing the attack we notice that there was no available portal to observe the impact of the attack. The portal we had been using to collect PLC data up to this step was ScadaBR, the HMI, which is now blocked from any updates due to the DoS attack on the PLC. Therefore we set up an alternate point of data collection on the OpenPLC-Simulink interface.

During the attack, we noticed that the true value of the height limit parameter would sporadically appear on ScadaBR. This is because the PLC was able to respond to the HML read request, revealing the true value of the parameter. Due to the host's limited processing power, increasing the amount of traffic causes the workstation virtual machine to crash. To achieve the full DoS effect, we decided to decrease the amount of memory allocated to the virtual machine which hosts the ICS router. Figure E.3 shows the true value of the PLC parameters and the value visible on ScadaBR. The attack also generates a large amount of traffic on the ICS network, as shown in figure E.1.

We concluded that the attack was **successful** in causing damage to the ICS. It successfully blocked the HMI operator from observing any PLC updates and it altered the height limit of the PLC. The height of the water which exceeds the normal limit would be damaging for a physical water tank with a volume limit.

5.3 Network Anomaly detection

In the previous section, we discussed how the attacks can be evaluated using different metrics. This method of evaluation requires knowledge of the type of attacks performed, in order to deduce suitable metrics. The relevant metrics will highlight the presence of an ongoing attack, which we can detect by sight. This approach to evaluating network state and detecting attacks is not feasible in the real world. Therefore, we need a more general way to profile the traffic and automate the detection of an ongoing attack. This section will discuss the profiling and evaluate the performance of the anomaly detection models on the ICS network.

5.3.1 Preparation and Profiling

The Shannon entropy (Shannon 1948) measures the average level of uncertainty in a set of data. In computer networks, it can be used as a metric to quantitatively profile a network. Shannon's entropy for a network feature distribution can be calculated using the following formula:

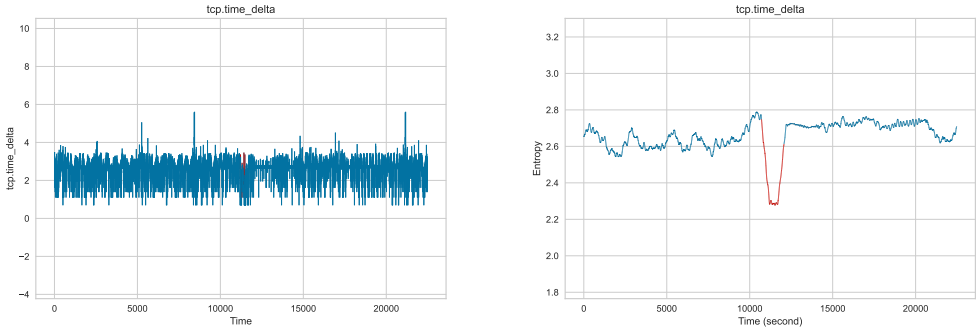
$$H = -\sum_{i=0}^{n} P(X_i) \ln P(X_i) \tag{5.1}$$

Where X is a set of unique possible values of the network feature and $P(X_i)$ is the probability of the ith value appearing in the network feature.

For example, assume that all packets in one PLC communication loop are being delivered over the network at time T = 0. This implies that we have 16 packets in that second, 8 packets with IP source address = PLC address and 8 packets with IP source address = HMI address. We will calculate the Shannon entropy of the IP source address. From the context given above, we deduce, X = PLC address, HMI address, P(PLC) = 0.5, and P(HMI) = 0.5. The Shannon entropy of the IP source address at T = 0 is then equal to $-[P(PLC) \ln P(PLC) + P(HMI) \ln P(HMI)] \approx 0.69$.

We will use the Shannon entropy to profile the traffic because it is capable of revealing the impact of each attack on the characteristic of network features, including impacts which might not be visible using the previous metrics. Because we are capturing the packets over time, we can profile the traffic by calculating the Shannon entropy of the feature distributions in a packet over time. Figure E.4 shows the preview of the resulting dataset.

An all-out zero entropy occurs at different points in time. This is because there could be only one packet present at a point in time, however rarely. The value in the packet will completely dominate the probability distribution (P(k) = 1). As a result, the entropy of all traffic features is zero. A sudden drop in all feature entropy will likely be falsely classified as an anomaly. Therefore, we added a step in preparation to remove the occasional lone packet.



(a) Raw entropy over time *(b) Convoluted entropy over time*

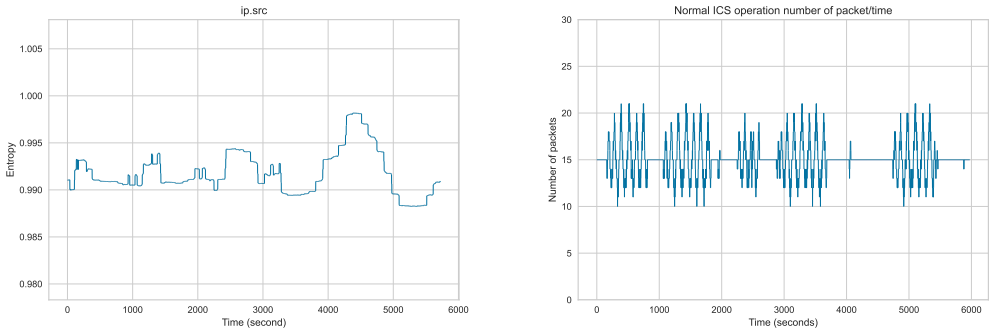
Figure 5.6: *The time delta entropy during ARP poisoning*

A convolution layer is added to modulate the entropy values. Figure 5.6a shows the raw time delta entropy value over time, where the lines highlighted in red represent the ongoing attack. The data has an overwhelming amount of noise, which decreases the performance of the anomaly detection model. Therefore we apply the convolution operator to the data using an evenly weighted kernel. The convolution operator in this evaluation is defined as:

$$H'_t = \sum_{m=0}^{w} \frac{H_(t-m)}{w} \tag{5.2}$$

Where w is the size of the convolution box.

The convolution operation filters out the noise while revealing the outlier behaviour, which resulted in a more defined pattern in traffic feature entropy. The difference in the data after the application of the convolution layer is shown in figure 5.6b. We can visibly see the presence of the attack, which was hidden in the noise.

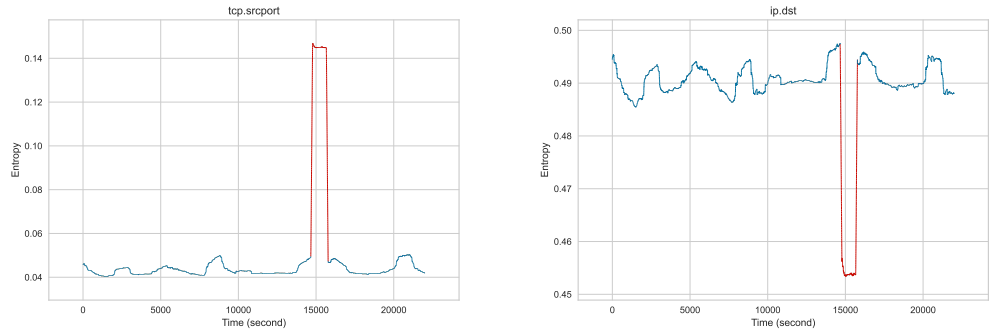


(a) IP source address entropy over time *(b) Baseline network packet over time*

Figure 5.7: *The baseline network entropy*

Figure 5.7a shows **the baseline IP source address entropy**. The baseline traffic contains only the communication between the PLC and the HMI. However, the feature entropy will still vary over time for the same reason the number of packets varies over time. We can see the relationship between the number of packets over time and entropy over time in figure 5.7. At times when there is a change in frequency, there is an increased amount of uncertainty, and therefore, increased entropy. A larger entropy value translates to larger randomness and unpredictability. A smaller entropy value translates to smaller randomness and unpredictability. Although the variability of the entropy as an effect of frequency change is very small, it will likely not exceed the anomaly detection model threshold.

Figure E.2 shows the **impact of reconnaissance** on the network entropy in its two most affected areas. We can observe a large and sudden increase in the entropy of the IP destination address and destination port. This is because reconnaissance generates traffic with more random ports and addresses.

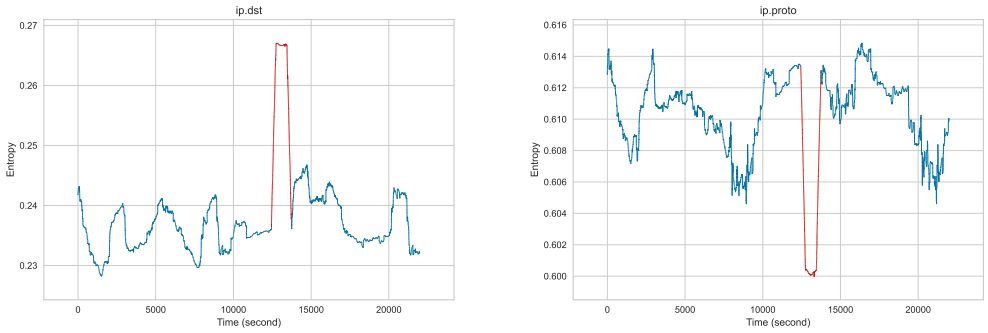


(a) Source port *(b) IP destination address*

Figure 5.8: *Impact of SYN-flooding attack on the network entropy*

The impact of the SYN flooding attack is shown in figure 5.8. During the attack, the IP

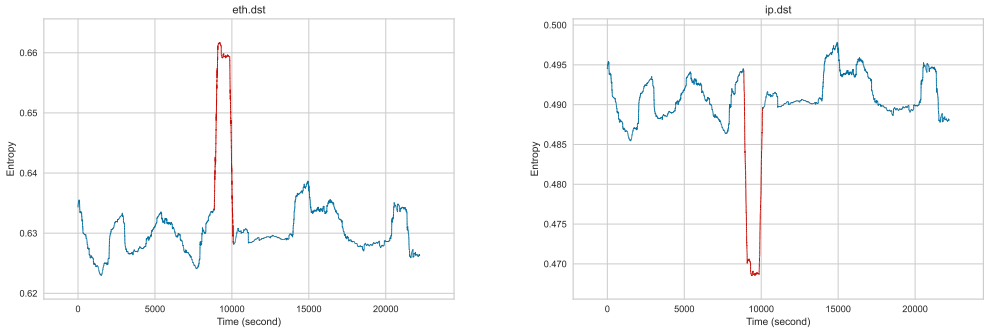
destination address which dominated the set is predictably the victim's address. Therefore, we see a decrease in entropy at the time of the attack in the IP destination address. On the other hand, figure 5.8a shows increased randomness in the source port, as a result of SYN packet flooding from all ports the attacker has available.



(a) IP destination address *(b) IP protocol*

Figure 5.9: *Impact of readbombing attack on the network entropy*

The readbombing attack has a similar effect to the SYN-flooding attack in the domain of the number of packets over time. However, the characteristics of both attacks are revealed to be different when profiled with entropy. During a readbombing attack, the PLC will attempt to reply to all of the attacker's requests. This increases the randomness of the IP destination address, while SYN-flooding decreases it. The protocol entropy during the time of the attack will also decrease. This is still similar to the SYN flooding attack (where TCP dominates the protocol distribution), but it is much less predictable. Readbombing attack is better at keeping the distribution of the protocol close to normal operation, it floods the PLC with valid read requests which will receive a valid response. However, the PLC cannot respond to all the requests and achieve normal protocol distribution. The read requests having Modbus protocol will still dominate the traffic at the time. The entropy of protocols decreases as a result of readbombing attack.



(a) Ethernet destination address *(b) IP destination address*

Figure 5.10: *Impact of ARP poisoning and injection attack on the network entropy*

Another characteristic that is revealed when profiling with entropy is the presence of **ARP poisoning and injection attack**. During an ARP poisoning attack, the Ethernet addresses in the packets are altered while the IP addresses stay the same. The number of packets over

time increases only slightly during the injection attack. The only noticeable indicator for ARP poisoning was a small spike where the attacker tries to collect all MAC addresses, seen in figure 5.4. However, a small change in the number of packets resulted in a slightly more significant change in the entropy. It creates a situation where one address appears more than the other (during re-transmission and re-connection), which resulted in a decrease in entropy. The biggest indication of ARP poisoning is the change in the relationship between the IP address and Ethernet address entropy. In a normal operation, an IP address and an Ethernet address have a direct relationship, because they are direct mapping of each other. But with the ARP cache spoofed, the IP addresses are mapped to the attacker's Ethernet address instead. Therefore, only the entropy of Ethernet addresses will be increased (figure 5.10), breaking the direct relationship between Ethernet address and IP address.

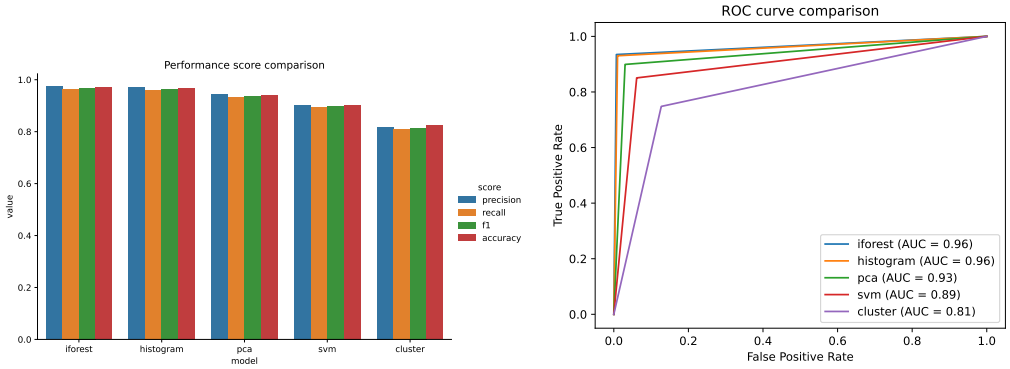
5.3.2 Anomaly detection models

The final step in anomaly detection is to evaluate the performance of different anomaly detection models over the data (**Aim 3**). We used the models mentioned in the background for their suitability with the datasets. In this section, we first evaluate the model's overall performance using the evaluation set containing all attacks. Then we evaluate the model's performance on each attack using sets of individual attacks.

The following features are used in the detection:

- Ethernet source address entropy
- Ethernet destination address entropy
- IP source address entropy
- IP destination address entropy
- Packet Length entropy
- Packet time-to-live entropy
- Protocol entropy

Figure 5.11a shows a comparison of model performance using four different performance metrics: Precision, Recall, Accuracy and F1 score. We also compared the ROC curve plotted with the prediction from each model. We concluded that isolation forest and histogram-based outlier detection models are the best at detecting overall attacks.



(a) Model performance on the evaluation set *(b) Receiver characteristic curve*

Figure 5.11: *Anomaly Detection models performance*

Although isolation forest and histogram-based outlier detection have the best performance overall, an investigation of models' performance over specific types of attacks reveals that support vector

machine and clustering outlier detection model achieved better performance in DoS type attack detection. The ROC curve and performance score comparisons of the model on each type of attack can be found in appendix F.2 and F.3.

5.4 Difficulties of malware-generated traffic detection

Following the attack chain in Section 4.3, we successfully downloaded the payload stager onto the workstation. The payload stager downloaded the MSFvenom payload which downloaded Meterpreter onto the workstation. We were able to gain a foothold in the system.

Table 5.1: *Result of the Windows exploits*

Vulnerability ID	Vulnerable	Privilege Escalated	Persistence	Successful
CVE-2021-40449	Yes	Yes	Yes	Yes
CVE-2022-21882	Yes	Yes	Yes	Yes
CVE-2022-21999	Yes	No	No	No

We performed the full attack three times with different exploitation techniques during privilege escalation. The workstation is vulnerable to all three exploits, but only two were successful at privilege escalation and persistence (Table 5.1). The final payload execution of SpoolFool exploitation (CVE-2022-21999) was blocked by the Windows Defender, and the attack was not successful.

The detection and evaluation of malware-generated traffic in the corporate network are done separately from the ICS network. Both networks have very different characteristics due to their different functionality.

Unfortunately, the profiling and detection methods used for the ICS network could not achieve the same level of performance on the corporate network. The following subsections will expand on the reason we failed to detect malware traffic on the corporate network.

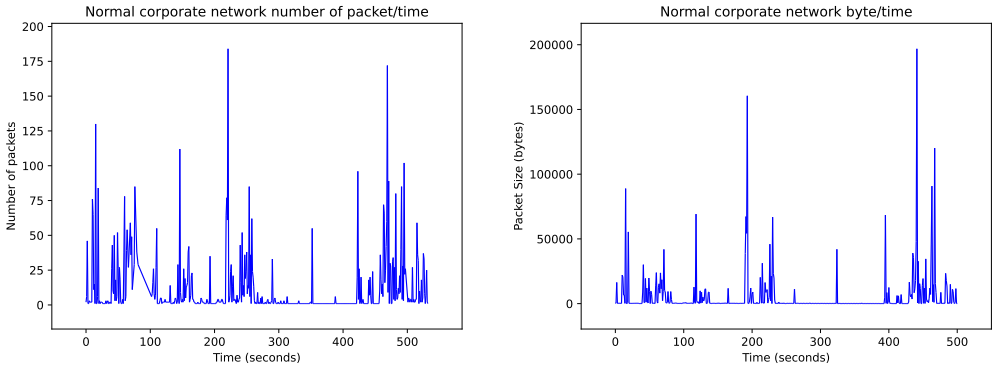
5.4.1 Malware or Application

We use the same metric for ICS on the corporate traffic, the number of packets over time metric. An additional metric is employed to quantify the impact of malware traffic on the network: the network bytes over time metric. Theoretically, the metrics should show some changes since the malware downloads malicious payloads and communicates with the attacker over the network.

We start by establishing **the baseline** using the two metrics. The baseline traffic contains traffic from different applications on Windows machines. Figure 5.12, shows the baseline network with extreme values filtered out as an attempt to reveal patterns. The baseline plot before filtering is in G.1. However, we cannot detect any pattern by sight.

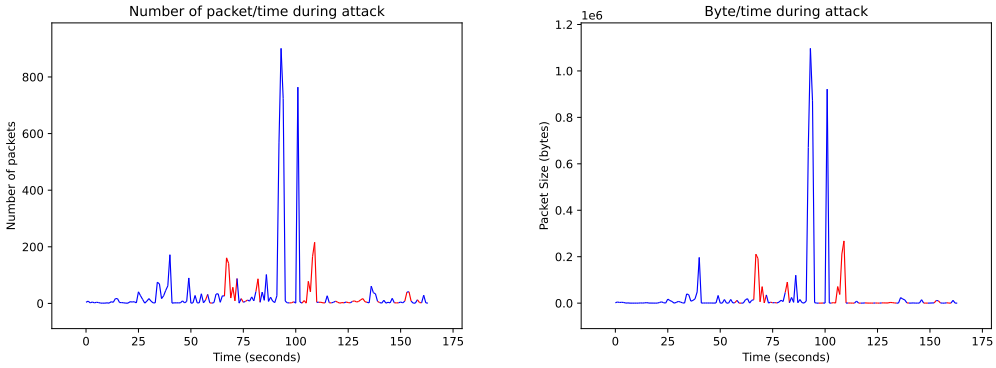
We use the metrics over the traffic generated by the two successful attacks. This produced figures 5.13 and 5.14. Both figures show very different results. In Figure 5.13, the malware traffic is indistinguishable from the application traffic, the malware produces even smaller changes than the Windows applications. The malware is creating more changes in the traffic than the Windows application in figure 5.14, however, this is only because of the timing of the attack. The malware traffic and application traffic remains indistinguishable in terms of the number of packets over time and network bytes over time.

We then calculate the corporate network entropy, which might reveal some hidden characteristics of the network. Unlike the ICS network, the corporate network does not seem to have any entropy patterns. This can be seen in figure G.2.



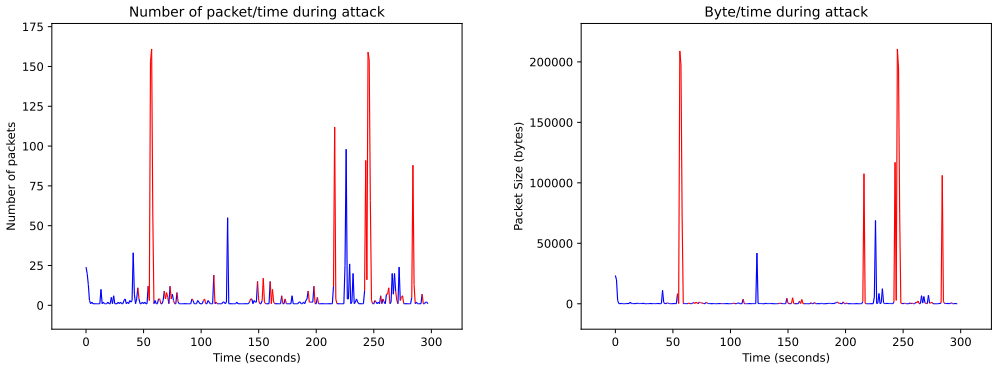
(a) Application traffic packet over time *(b) Application traffic byte over time*

Figure 5.12: *Normal corporate network traffic characteristic*



(a) Malware traffic packet over time *(b) Malware traffic byte over time*

Figure 5.13: *Corporate traffic during CVE-2021-40449 exploitation*



(a) Malware traffic packet over time *(b) Malware traffic byte over time*

Figure 5.14: *Corporate traffic during CVE-2022-21882 exploitation*

In view of the arguments above, we conclude that our method of profiling the traffic is unsuitable for corporate network traffic. This is due to the similarity of traffic generated by a legitimate

application to the traffic generated by malware. Both types of traffic occur at a random time, both cause an instantaneous increase in the traffic and contain large amounts of data.

5.4.2 Anomaly detection model performance

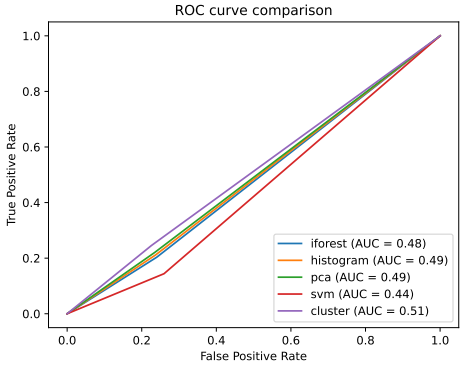


Figure 5.15: *Model ROC curve*

The anomaly detection models are trained with the corporate network data, collected separately from the ICS network. We evaluate the models using a set of individual exploits concatenated together. The same metrics are used to measure the performance of the model on the corporate network dataset. Figure 5.15 shows that all of the models performed no better than a random classifier. Therefore they are absolutely inefficient in malware traffic detection.

5.4.3 Alternative approach

Malware traffic detection has always been challenging for IDS researchers for the same reason it is challenging in this experiment. It generates lightweight traffic that does not disrupt the network, unlike volume-based attacks such as DDoS. We have proven the detection approach taken in this experiment to be inefficient against malware traffic. We must now look for an alternative approach which could be applied in the future.

A malware traffic detection tool by Yen and Reiter (2008) was able to detect 87.5% of botnet malware traffic. Their approach utilises flow aggregate based on three different features: destination, payload, and platform. In destination-based aggregation, they focus on the flow between internal hosts and external hosts. An external host which has more than usual interaction with the internal host compared to the baseline is marked suspicious.

Assuming that malware-generated traffic will have different payload byte frequency distribution than normal traffic, they aggregated flows based on the edit distance with substring moves, grouping flow based on the distance between payloads. The last aggregation rule is the platform, this refers to the operating system of the hosts in the communication. They identify the platform by using fingerprinting rules, derived from the characteristic of each operating system in managing a packet.

6 Conclusion

In relation to the aims and objectives of this study, we developed a virtual testbed as a secure environment for simulating ICS and ICS attacks. Then, we implemented an attack for the virtual testbed and analysed the impact of the attacks focusing on the ICS. Finally, we present a method for ICS attack detection and evaluate its effectiveness with data collected from the testbed.

We analysed the virtual testbed requirements and developed the testbed using Virtual Machine and container technologies. The components of the testbed are simulated using established tools such as Simulink, OpenPLC, ScadaBR and Kali Linux. We display that the testbed is effective in simulating the behaviour of an ICS and exhibiting behaviour change during the attack. However, this includes limitations; The testbed lacks complexity, and has a low number of components and interactions, due to the limited capacity of the host machine.

The attacks in this research are designed and implemented with respect to the MITRE ATT&CK Framework for Windows and the MITRE ATT&CK Framework for ICS for generalisability. We performed the attack on the virtual testbed and collected results over the network and the PLC. The attack was successful in compromising the workstation and damaging the ICS. The volume-based attacks and injection attack were effective on the ICS network. However, the injection attack has flaws due to the lack of consideration for the rate of physical process operation. Being considerate of the physical aspects when designing the attack will improve these flaws.

We implemented an attack detection method by using machine learning models on the Shannon entropy of the network feature distributions. Shannon's entropy showed us the characteristics of each attack compared to the baseline network entropy. Our anomaly detection models were proven effective on ICS network data, the best performing models were Isolation forest and Clustering-based anomaly detection models, both achieving the AUC score of 0.96. On the other hand, the profiling method and the anomaly detection models were proven ineffective on the corporate network data, containing application traffic and malware traffic. Due to similarities between both types of traffic, the entropy of the network distribution during normal operation and during attacks is indistinguishable. We suggest alternative profiling techniques that could be used for corporate network malware traffic detection.

6.1 Future Work

In relation to the success and limitations of the virtual testbed, the attacks performed and the attack detection method mentioned above, we propose future work for each component.

Due to the simplicity of the virtual testbed, the attack did have an explicit step in identifying devices on the network and their roles, a step which is usually taken in real-life attacks. The virtual testbed can be improved by adding more PLCs, HMIs, workstations and other ICS components to increase the complexity and the number of interactions of the components in the network. We could integrate PLCs from different manufacturers to increase the variety of the protocols in the testbed. This would give us more COTS OS devices in the ICS network and allow us to study the vulnerabilities of commercial PLCs. The physical process could be switched to a more complex process, such as the Tennessee Eastman process, which is suitable for multiple PLCs and

can be modelled using Simulink. This would result in a system that is more realistic, allowing for the development of more sophisticated attacks on the ICS network.

In the reconnaissance step of the attack chain, we naively choose one variable to be our target without performing an analysis of the potential impact. In the future, we should collect the values of all variables for a longer interval of time. Then we use data processing techniques to find correlations and establish relationships between the variables. It would also be interesting to develop a replication strategy for the malware since we cannot always rely on the ICS workers to mistakenly download the payload.

The anomaly detection method was effective in detecting attacks in the data gathered from the ICS network in this experiment. However, to evaluate whether the method is suitable for anomaly detection in a real ICS network, it should be tested with different ICS datasets to measure its accuracy and scalability. The model could then be deployed over an ICS testbed to measure its performance in real time. In such cases, we would have to take into consideration the real-time sampling method and the trade-off between cost and accuracy.

We could improve malware traffic detection by using different techniques in profiling traffic, such as flow aggregates, as mentioned above. We could also approach the problem using signal processing and decompose the signal into its constituent frequencies in order to reveal network abnormalities, created by the presence of malware.

A Virtual Testbed Architecture

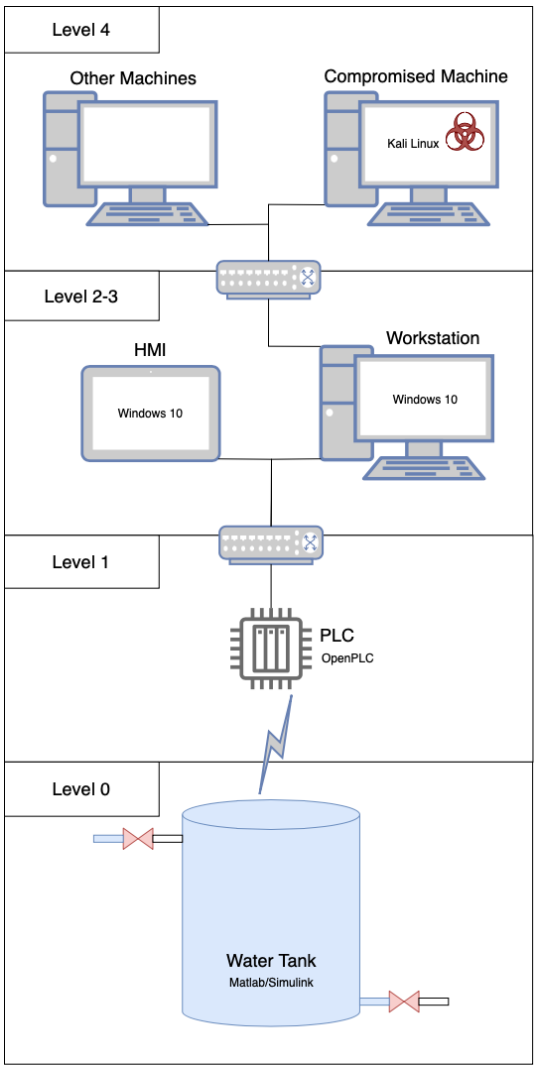


Figure A.1: The virtual testbed architecture

B Screenshots of the Virtual Testbed

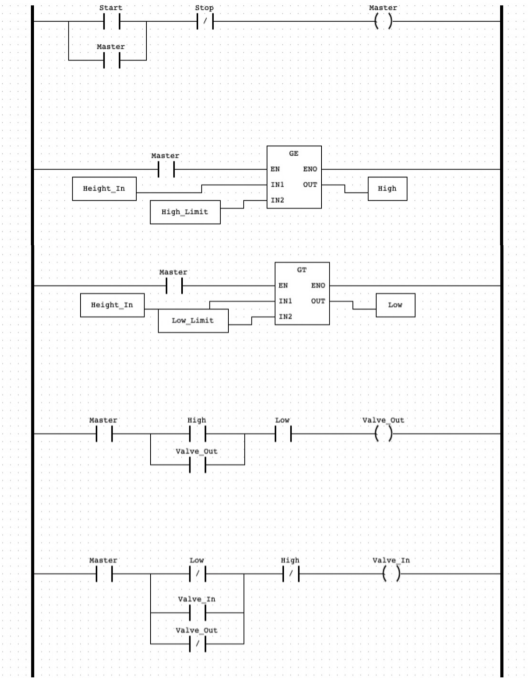


Figure B.1: *The PLC ladder logic program*

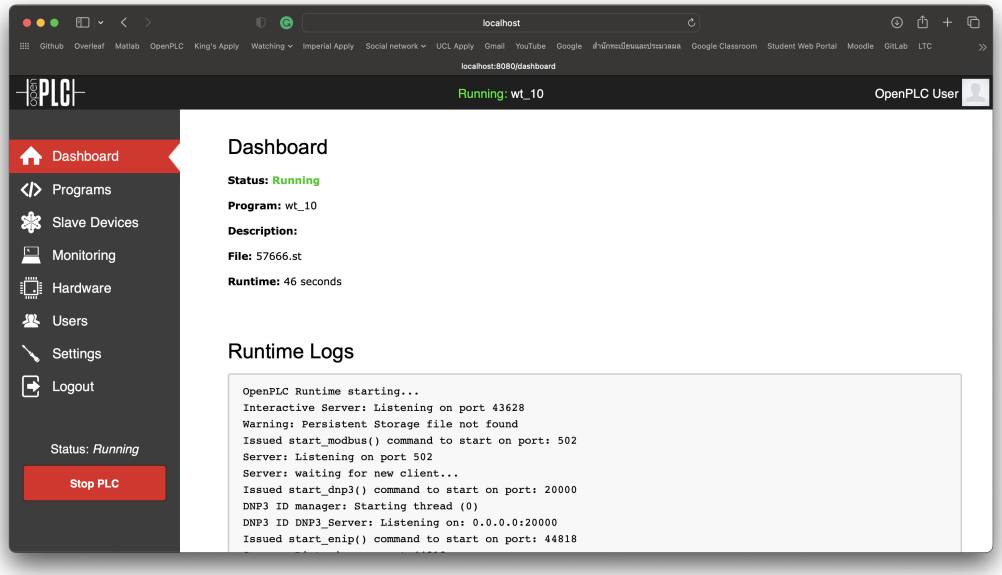


Figure B.2: *The OpenPLC webserver*

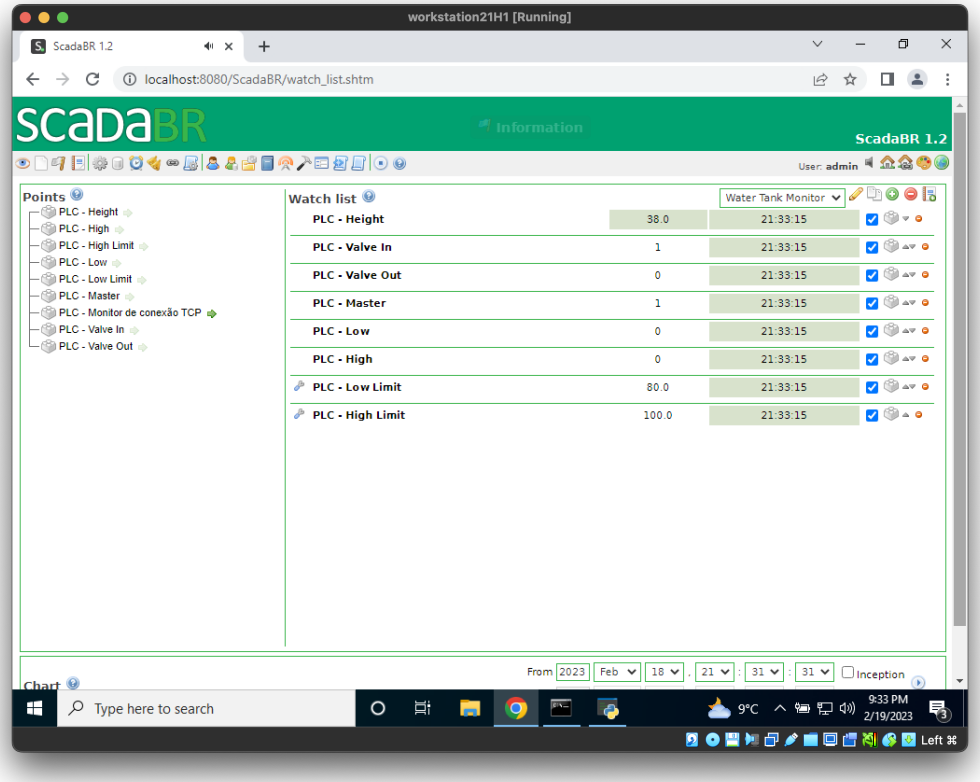


Figure B.3: *ScadaBR variable watch list*

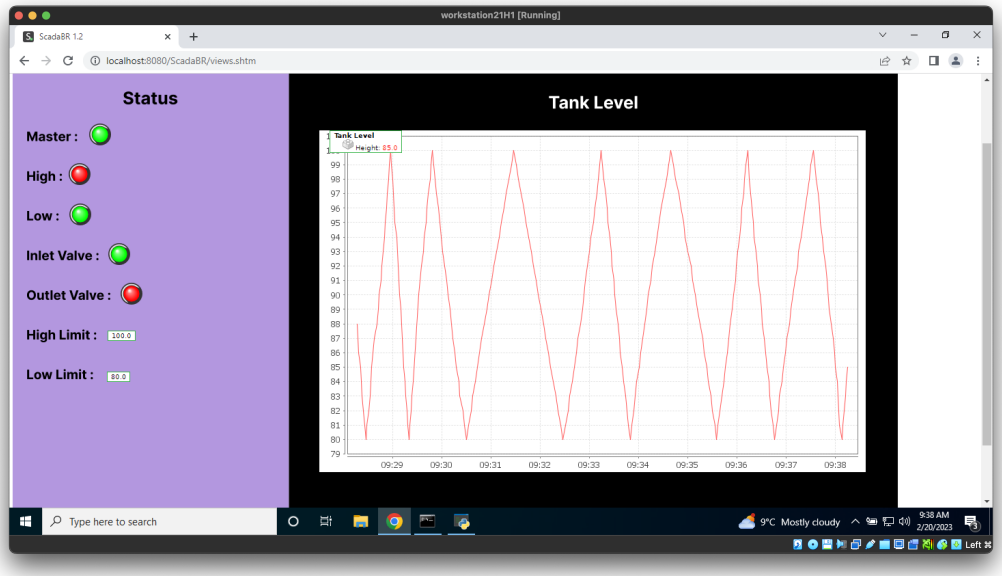


Figure B.4: The graphical view in ScadaBR

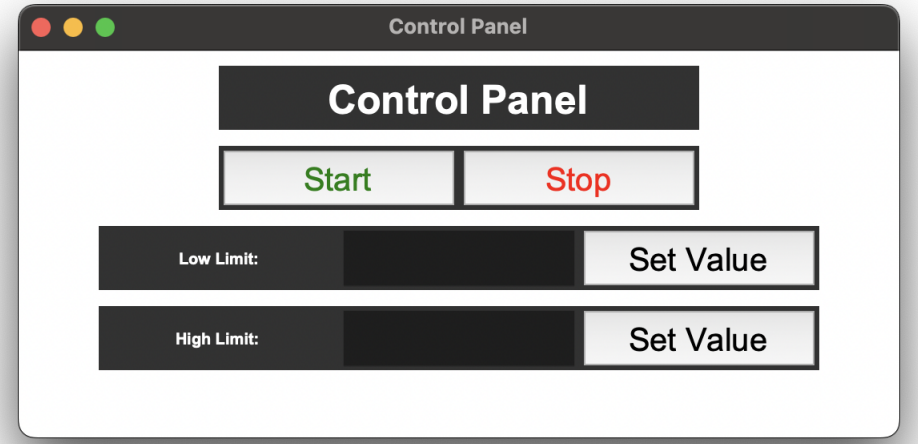


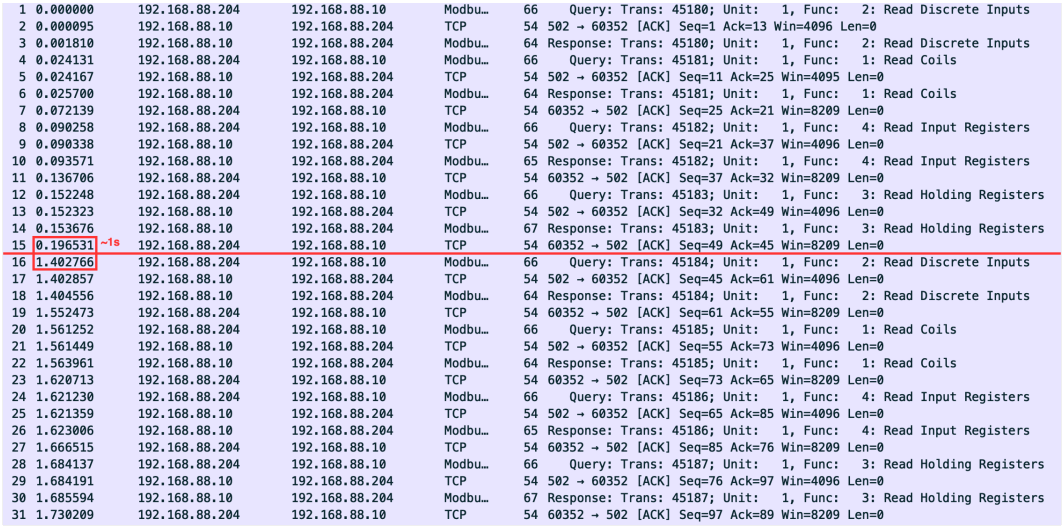
Figure B.5: The control panel

C Mapping to the MITRE ATT&CK Framework

Table C.1: *Mapping to the MITRE ATT&CK Framework*

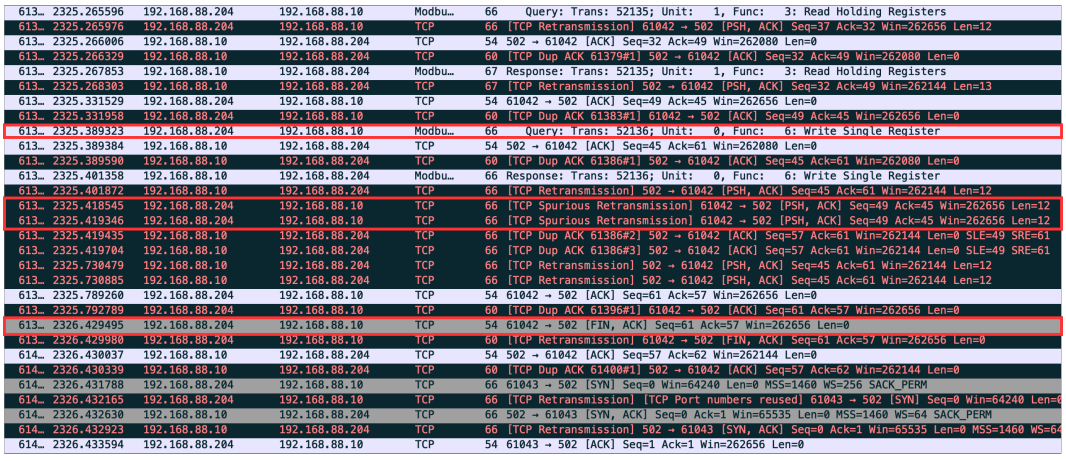
Tactic	**ID**	**Reference**
Reconnaissance	T1595 T1040	Lyon (2009) Williams (2023)
Resource Development	T1587 T1588	Kennedy et al. (2011)
Initial Access	T1534 T0866 T0865	Alladi et al. (2020) Wen et al. (2013)
Execution	T1204.002 T1059.001 T1059.003	Abraham and Chengalur-Smith (2010) Kumar et al. (2020)
Persistence	T1547.001 T1136.001	Sharma et al. (2022) Berba (2021)
Privilege Escalation	T1543 T1055.001 T1068	Kc et al. (2003) Stephenfewer Matrosov et al. (2010)
Defence Evasion	T1027.004 T1055.001 T1562 T1620 T01564.003	You and Yim (2010) Stephenfewer Ortiz (2020a) Danielbohannon
Credential Access	T1003	Rahalkar and Jaswal
Discovery	T1069 T1040	Archiveddocs Williams (2023)
Command and Control	T1219 T1095	Huc (2023) Atwell et al. (2016)
Collection	T0830	Nachreiner (2003)
Inhibit Response Function	T0814	Govil et al. (2018)
Impair Process Control	T0836 T0855	Morris and Gao (2013)
Impact	T0826 T0879 T0882	Cárdenas et al. (2011)

D Screenshots from Attack Implementation



1	0.000000	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45180; Unit: 1, Func: 2: Read Discrete Inputs
2	0.000095	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=1 Ack=13 Win=4096 Len=0
3	0.001810	192.168.88.10	192.168.88.204	Modbu…	64	Response: Trans: 45180; Unit: 1, Func: 2: Read Discrete Inputs
4	0.024131	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45181; Unit: 1, Func: 1: Read Coils
5	0.024167	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=11 Ack=25 Win=4095 Len=0
6	0.025700	192.168.88.10	192.168.88.204	Modbu…	64	Response: Trans: 45181; Unit: 1, Func: 1: Read Coils
7	0.072139	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=25 Ack=21 Win=8209 Len=0
8	0.090258	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45182; Unit: 1, Func: 4: Read Input Registers
9	0.090338	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=21 Ack=37 Win=4096 Len=0
10	0.093571	192.168.88.10	192.168.88.204	Modbu…	65	Response: Trans: 45182; Unit: 1, Func: 4: Read Input Registers
11	0.136706	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=37 Ack=32 Win=8209 Len=0
12	0.152248	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45183; Unit: 1, Func: 3: Read Holding Registers
13	0.152323	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=32 Ack=49 Win=4096 Len=0
14	0.153676	192.168.88.10	192.168.88.204	Modbu…	67	Response: Trans: 45183; Unit: 1, Func: 3: Read Holding Registers
15	0.196531 ~1s	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=49 Ack=45 Win=8209 Len=0
16	1.402766	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45184; Unit: 1, Func: 2: Read Discrete Inputs
17	1.402857	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=45 Ack=61 Win=4096 Len=0
18	1.404556	192.168.88.10	192.168.88.204	Modbu…	64	Response: Trans: 45184; Unit: 1, Func: 2: Read Discrete Inputs
19	1.552473	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=61 Ack=55 Win=8209 Len=0
20	1.561252	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45185; Unit: 1, Func: 1: Read Coils
21	1.561449	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=55 Ack=73 Win=4096 Len=0
22	1.563961	192.168.88.10	192.168.88.204	Modbu…	64	Response: Trans: 45185; Unit: 1, Func: 1: Read Coils
23	1.620713	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=73 Ack=65 Win=8209 Len=0
24	1.621230	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45186; Unit: 1, Func: 4: Read Input Registers
25	1.621359	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=65 Ack=85 Win=4096 Len=0
26	1.623006	192.168.88.10	192.168.88.204	Modbu…	65	Response: Trans: 45186; Unit: 1, Func: 4: Read Input Registers
27	1.666515	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=85 Ack=76 Win=8209 Len=0
28	1.684137	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 45187; Unit: 1, Func: 3: Read Holding Registers
29	1.684191	192.168.88.10	192.168.88.204	TCP	54	502 → 60352 [ACK] Seq=76 Ack=97 Win=4096 Len=0
30	1.685594	192.168.88.10	192.168.88.204	Modbu…	67	Response: Trans: 45187; Unit: 1, Func: 3: Read Holding Registers
31	1.730209	192.168.88.204	192.168.88.10	TCP	54	60352 → 502 [ACK] Seq=97 Ack=89 Win=8209 Len=0

Figure D.1: *The PLC communication loop showing one-second window between loops*



613…	2325.265596	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 52135; Unit: 1, Func: 3: Read Holding Registers
613…	2325.265976	192.168.88.204	192.168.88.10	TCP	66	[TCP Retransmission] 61042 → 502 [PSH, ACK] Seq=37 Ack=32 Win=262656 Len=12
613…	2325.266006	192.168.88.10	192.168.88.204	TCP	54	502 → 61042 [ACK] Seq=32 Ack=49 Win=262080 Len=0
613…	2325.266329	192.168.88.10	192.168.88.204	TCP	60	[TCP Dup ACK 61379#1] 502 → 61042 [ACK] Seq=32 Ack=49 Win=262080 Len=0
613…	2325.267853	192.168.88.10	192.168.88.204	Modbu…	67	Response: Trans: 52135; Unit: 1, Func: 3: Read Holding Registers
613…	2325.268303	192.168.88.10	192.168.88.204	TCP	67	[TCP Retransmission] 502 → 61042 [PSH, ACK] Seq=32 Ack=49 Win=262144 Len=13
613…	2325.331529	192.168.88.204	192.168.88.10	TCP	54	61042 → 502 [ACK] Seq=49 Ack=45 Win=262656 Len=0
613…	2325.331958	192.168.88.204	192.168.88.10	TCP	60	[TCP Dup ACK 61383#1] 61042 → 502 [ACK] Seq=49 Ack=45 Win=262656 Len=0
613…	2325.389323	192.168.88.204	192.168.88.10	Modbu…	66	Query: Trans: 52136; Unit: 0, Func: 6: Write Single Register
613…	2325.389384	192.168.88.10	192.168.88.204	TCP	54	502 → 61042 [ACK] Seq=45 Ack=61 Win=262080 Len=0
613…	2325.389590	192.168.88.10	192.168.88.204	TCP	60	[TCP Dup ACK 61386#1] 502 → 61042 [ACK] Seq=45 Ack=61 Win=262080 Len=0
613…	2325.401358	192.168.88.10	192.168.88.204	Modbu…	66	Response: Trans: 52136; Unit: 0, Func: 6: Write Single Register
613…	2325.401872	192.168.88.10	192.168.88.204	TCP	66	[TCP Retransmission] 502 → 61042 [PSH, ACK] Seq=45 Ack=61 Win=262144 Len=12
613…	2325.418545	192.168.88.204	192.168.88.10	TCP	66	[TCP Spurious Retransmission] 61042 → 502 [PSH, ACK] Seq=49 Ack=45 Win=262656 Len=12
613…	2325.419346	192.168.88.204	192.168.88.10	TCP	66	[TCP Spurious Retransmission] 61042 → 502 [PSH, ACK] Seq=49 Ack=45 Win=262656 Len=12
613…	2325.419435	192.168.88.10	192.168.88.204	TCP	66	[TCP Dup ACK 61386#2] 502 → 61042 [ACK] Seq=57 Ack=61 Win=262144 Len=0 SLE=49 SRE=61
613…	2325.419704	192.168.88.10	192.168.88.204	TCP	66	[TCP Dup ACK 61386#3] 502 → 61042 [ACK] Seq=57 Ack=61 Win=262144 Len=0 SLE=49 SRE=61
613…	2325.730479	192.168.88.10	192.168.88.204	TCP	66	[TCP Retransmission] 502 → 61042 [PSH, ACK] Seq=45 Ack=61 Win=262144 Len=12
613…	2325.730885	192.168.88.10	192.168.88.204	TCP	66	[TCP Retransmission] 502 → 61042 [PSH, ACK] Seq=45 Ack=61 Win=262144 Len=12
613…	2325.789260	192.168.88.204	192.168.88.10	TCP	54	61042 → 502 [ACK] Seq=61 Ack=57 Win=262656 Len=0
613…	2325.792789	192.168.88.204	192.168.88.10	TCP	60	[TCP Dup ACK 61396#1] 61042 → 502 [ACK] Seq=61 Ack=57 Win=262656 Len=0
613…	2326.429495	192.168.88.204	192.168.88.10	TCP	54	61042 → 502 [FIN, ACK] Seq=61 Ack=57 Win=262656 Len=0
613…	2326.429980	192.168.88.204	192.168.88.10	TCP	60	[TCP Retransmission] 61042 → 502 [FIN, ACK] Seq=61 Ack=57 Win=262656 Len=0
614…	2326.430037	192.168.88.10	192.168.88.204	TCP	54	502 → 61042 [ACK] Seq=57 Ack=62 Win=262144 Len=0
614…	2326.430339	192.168.88.10	192.168.88.204	TCP	60	[TCP Dup ACK 61400#1] 502 → 61042 [ACK] Seq=57 Ack=62 Win=262144 Len=0
614…	2326.431788	192.168.88.204	192.168.88.10	TCP	66	61043 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
614…	2326.432165	192.168.88.204	192.168.88.10	TCP	66	[TCP Retransmission] [TCP Port numbers reused] 61043 → 502 [SYN] Seq=0 Win=64240 Len=0
614…	2326.432630	192.168.88.10	192.168.88.204	TCP	66	502 → 61043 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64 SACK_PERM
614…	2326.432923	192.168.88.10	192.168.88.204	TCP	66	[TCP Retransmission] 502 → 61043 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=64
614…	2326.433594	192.168.88.204	192.168.88.10	TCP	54	61043 → 502 [ACK] Seq=1 Ack=1 Win=262656 Len=0

Figure D.2: *Behaviour of the ICS network during packet injection. Highlighted frames from top to bottom: the injected packet, the discarded legitimate packet, the connection terminates*

E ICS Network Attacks

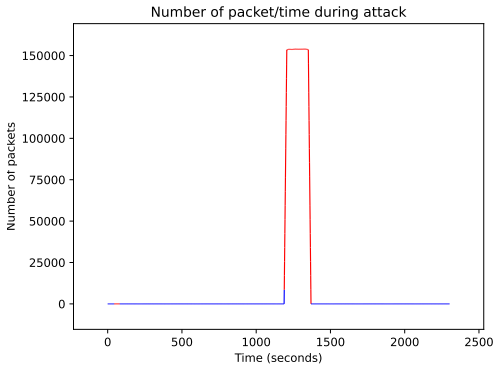
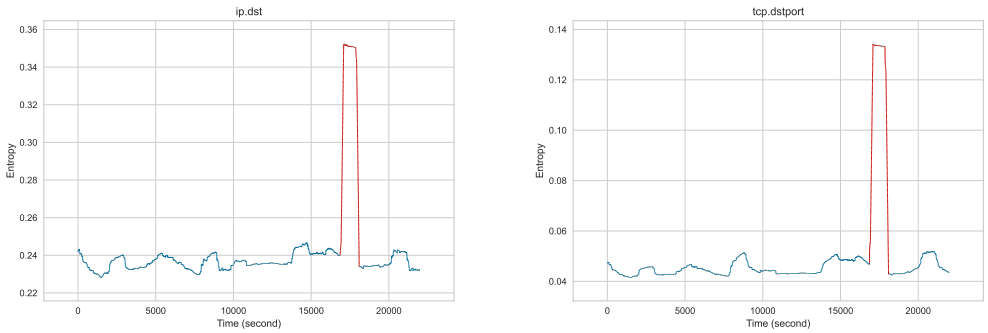
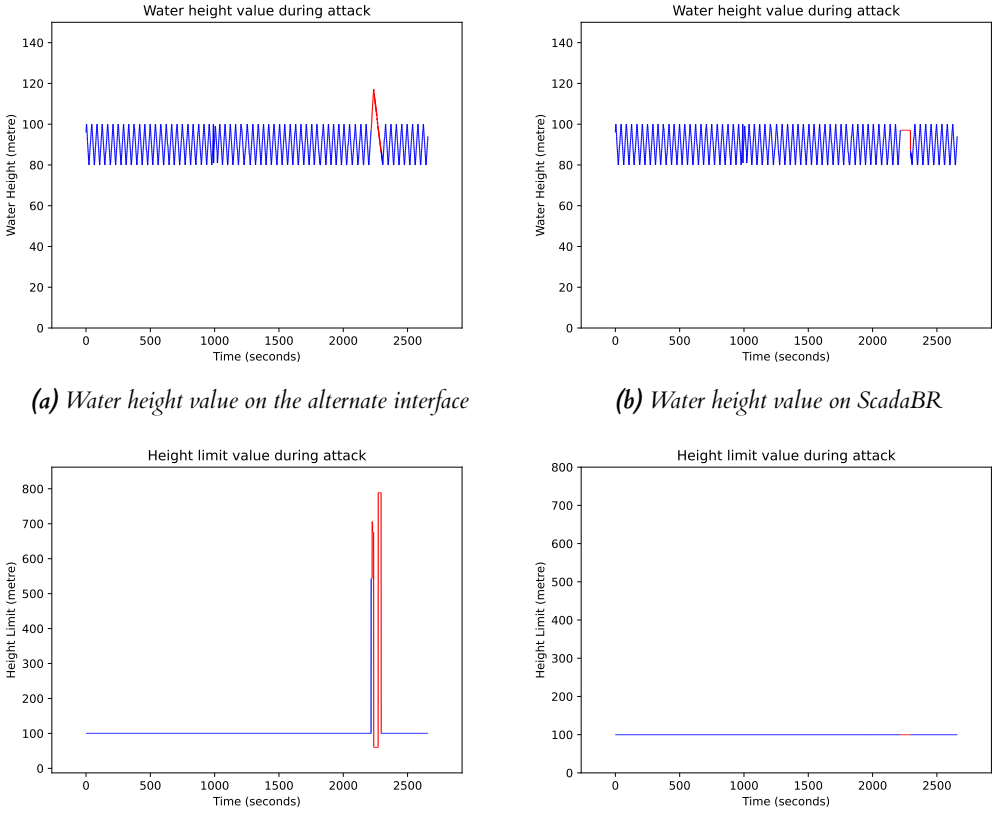


Figure E.1: *Impact of combination attack on the ICS network*



(a) *IP destination address* ***(b)*** *Destination port*

Figure E.2: *Impact of reconnaissance on the network entropy*



(a) Water height value on the alternate interface

(b) Water height value on ScadaBR

(c) Height limit value on the alternate interface

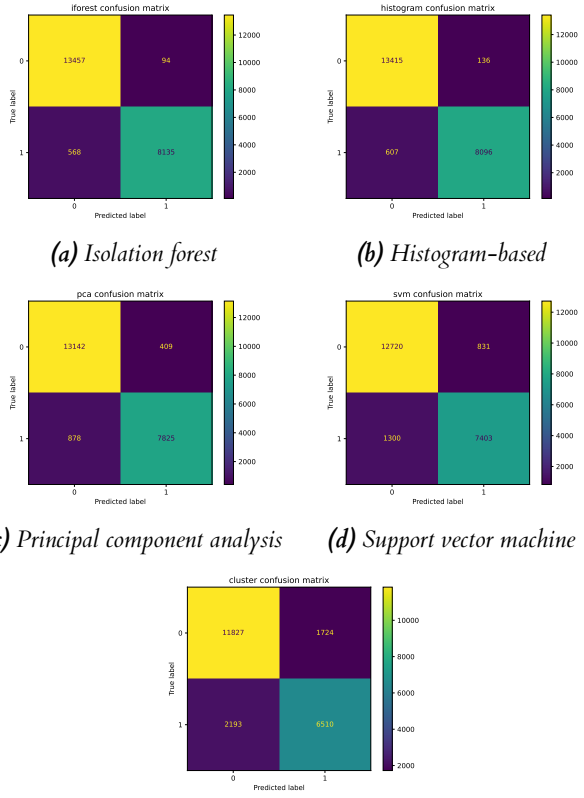
(d) Height limit value on ScadaBR

Figure E.3: *Impact of combination attack on the PLC*

time	eth.src	eth.dst	ip.src	ip.dst	ip.len	ip.id	ip.ttl	ip.proto	tcp.srcport	tcp.dstport	tcp.seq	tcp.ack	tcp.flags	tcp.window_	tcp.time_del
0	0.6829081	0.6829081	0.6829081	0.6829081	1.37605529	1.45080502	0.6829081	0.6829081	0.6829081	0.6829081	1.90853528	2.04493117	0.6829081	0.89820534	2.63905733
1	0.69141608	0.69141608	0.69141608	0.69141608	1.26224317	1.85465262	0.69141608	0.69141608	0.69141608	0.69141608	2.18083953	2.18083953	0.69141608	0.69141608	2.83321334
2	0.69246115	0.69246115	0.69246115	0.69246115	1.29964485	1.92743677	0.69246115	0.69246115	0.69246115	0.69246115	2.4743291	2.4743291	0.69246115	0.82588563	3.29583687
3	0.6917615	0.6917615	0.6917615	0.6917615	1.33418637	1.73255209	0.6917615	0.6917615	0.6917615	0.6917615	2.36073609	2.36073609	0.6917615	1.16436451	2.94443898
4	0.69246115	0.69246115	0.69246115	0.69246115	1.29964485	1.92743677	0.69246115	0.69246115	0.69246115	0.69246115	2.50629377	2.57701757	0.69246115	0.82588563	3.29583687
5	0.6917615	0.6917615	0.6917615	0.6917615	1.33418637	1.73255209	0.6917615	0.6917615	0.6917615	0.6917615	2.36073609	2.36073609	0.6917615	0.8628578	2.94443898
6	0.6859298	0.6859298	0.6859298	0.6859298	1.34168819	1.74100372	0.6859298	0.6859298	0.6859298	0.6859298	2.53252461	2.49800276	0.6859298	0.91559494	3.21887582
7	0.69314718	0.69314718	0.69314718	0.69314718	1.30345081	1.84443973	0.69314718	0.69314718	0.69314718	0.69314718	2.37189981	2.37189981	0.69314718	1.1921945	2.99573227
8	0.8218065	0.69059399	0.8218065	0.69059399	1.40730506	2.01267585	0.8218065	0.8218065	0.8218065	0.8218065	2.6203699	2.68856784	0.8218065	0.8218065	3.33220451
9	0.69314718	0.69314718	0.69314718	0.69314718	1.30345081	1.84443973	0.69314718	0.69314718	0.69314718	0.69314718	2.37189981	2.37189981	0.69314718	0.85568867	2.99573227
10	0.68900924	0.68900924	0.68900924	0.68900924	1.30369767	1.73563883	0.68900924	0.68900924	0.68900924	0.68900924	2.46090865	2.46090865	0.68900924	0.84546523	3.09104245
11	0.69234697	0.69234697	0.69234697	0.69234697	1.28872456	2.02612063	0.69234697	0.69234697	0.69234697	0.69234697	2.55345453	2.55345453	0.69234697	1.01331269	3.16342405
12	0.83789723	0.68461628	0.83789723	0.68461628	1.42586008	1.83902118	0.83789723	0.83789723	0.83789723	0.83789723	2.53275754	2.53275754	0.83789723	0.98755078	3.13549422
13	0.69314718	0.69314718	0.69314718	0.69314718	1.31437384	1.93560051	0.69314718	0.69314718	0.69314718	0.69314718	2.54266891	2.54266891	0.69314718	0.83656517	3.17805383
14	0.68900924	0.68900924	0.68900924	0.68900924	1.30369767	1.73563883	0.68900924	0.68900924	0.68900924	0.68900924	2.52392203	2.52392203	0.68900924	0.9347699	3.09104245
15	0.69314718	0.69314718	0.69314718	0.69314718	1.31437384	1.93560051	0.69314718	0.69314718	0.69314718	0.69314718	2.54266891	2.54266891	0.69314718	0.83656517	3.17805383

Figure E.4: *The Shannon entropy of the feature distributions in a packet over time*

F | ICS Anomaly Detection Model Comparisons



(a) Isolation forest

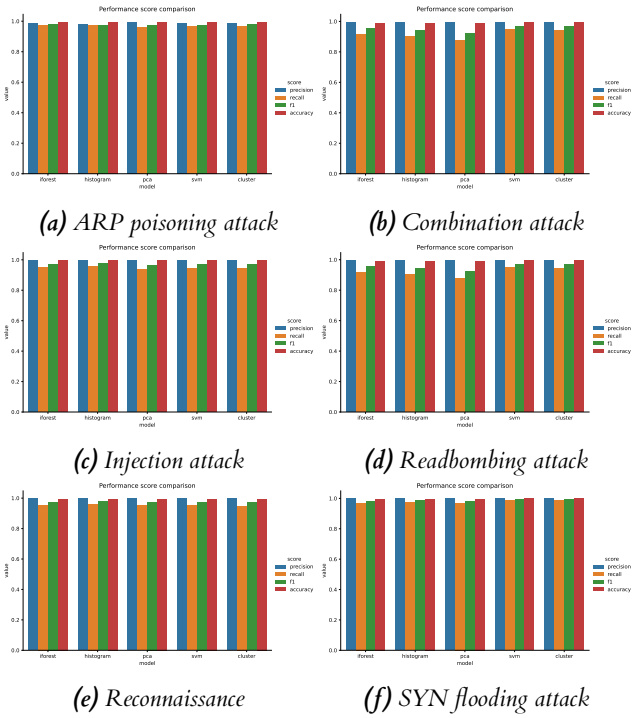
(b) Histogram-based

(c) Principal component analysis

(d) Support vector machine

(e) Clustering-based

Figure F.1: *Model confusion metrics*

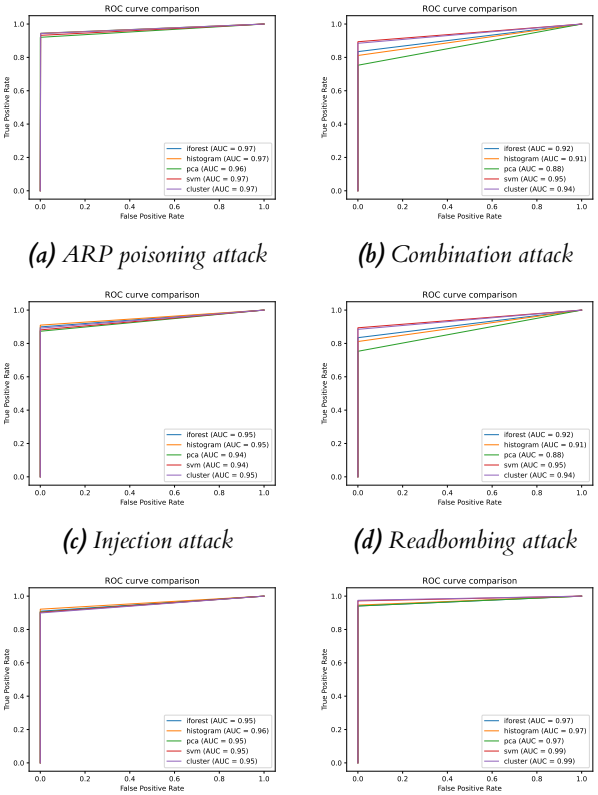


(a) ARP poisoning attack *(b) Combination attack*

(c) Injection attack *(d) Readbombing attack*

(e) Reconnaissance *(f) SYN flooding attack*

Figure F.2: *Model performance score comparison on different types of attacks*



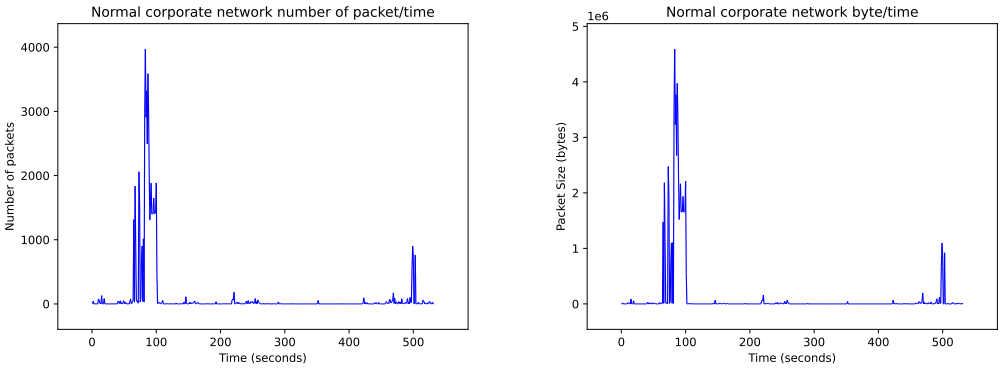
(a) ARP poisoning attack *(b) Combination attack*

(c) Injection attack *(d) Readbombing attack*

(e) Reconnaissance *(f) SYN flooding attack*

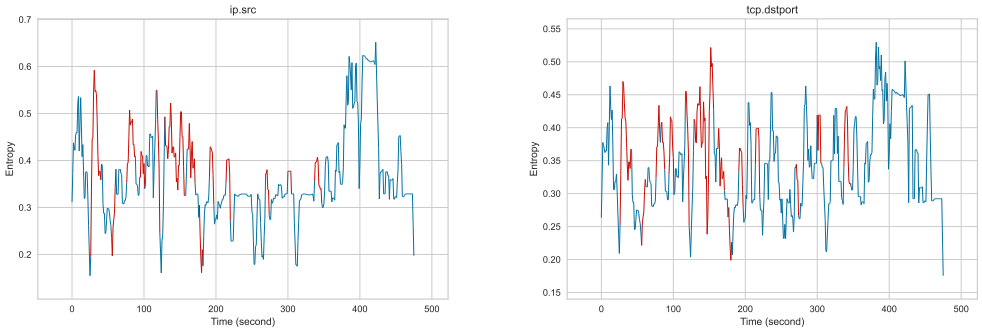
Figure F.3: *Model ROC curve on different types of attacks*

G Corporate Network Attacks



(a) Application traffic packet over time *(b) Application traffic byte over time*

Figure G.1: *Normal corporate network traffic characteristic*



(a) IP source address *(b) Destination port*

Figure G.2: *Impact of malware-generated traffic on the network entropy*

6 Bibliography

S. Abraham and I. Chengalur-Smith. An overview of social engineering malware: Trends, tactics, and implications. *Technology in Society*, 32(3):183–196, 2010.

C. M. Ahmed, V. R. Palleti, and A. P. Mathur. Wadi: a water distribution testbed for research in the design of secure cyber physical systems. In *Proceedings of the 3rd international workshop on cyber-physical systems for smart water networks*, pages 25–28, 2017.

T. Alladi, V. Chamola, and S. Zeadally. Industrial control systems: Cyberattack trends and countermeasures. *Computer Communications*, 155:1–8, 2020.

M. S. Almas, L. Vanfretti, S. Løvlund, and J. Gjerde. Open source scada implementation and pmu integration for power system monitoring and control applications. In *2014 IEEE PES General Meeting| Conference & Exposition*, pages 1–5. IEEE, 2014.

W. Alsabbagh, S. Amogbonjaye, D. Urrego, and P. Langendörfer. A stealthy false command injection attack on modbus based scada systems.

T. Alves. Thiagoralves/openplc_simulink-interface: Simulink interface program for openplc. URL `https://github.com/thiagoralves/OpenPLC_Simulink-Interface`.

T. Alves and T. Morris. Openplc: An iec 61,131–3 compliant open source industrial controller for cyber security research. *Computers & Security*, 78:364–379, 2018.

T. R. Alves, M. Buratto, F. M. De Souza, and T. V. Rodrigues. Openplc: An open source alternative to automation. In *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, pages 585–589. IEEE, 2014.

Archiveddocs. Net user. URL `https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2012-r2-and-2012/cc771865(v=ws.11)`.

R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He. Fuzziness based semi-supervised learning approach for intrusion detection system. *Information sciences*, 378:484–497, 2017.

C. Atwell, T. Blasi, and T. Hayajneh. Reverse tcp and social engineering attacks in the era of big data. In *2016 IEEE 2nd International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 90–95. IEEE, 2016.

S. M. H. Bamakan, H. Wang, T. Yingjie, and Y. Shi. An effective intrusion detection framework based on mclp/svm optimized by time-varying chaos particle swarm optimization. *Neurocomputing*, 199:90–102, 2016.

S. Behal and K. Kumar. Detection of ddos attacks and flash events using novel information theory metrics. *Computer Networks*, 116:96–110, 2017.

P. Berba. Hunting for persistence in linux (part 2): Account creation and manipulation, Nov 2021.

P. Bereziński, B. Jasiul, and M. Szpyrka. An entropy-based network anomaly detection method. *Entropy*, 17(4):2367–2408, 2015.

H. Bostani and M. Sheikhan. Hybrid of anomaly-based and specification-based ids for internet of things using unsupervised opf based on mapreduce approach. *Computer Communications*, 98: 52–71, 2017.

V. Bukhtoyarov and V. Zhukov. Ensemble-distributed approach in classification problem solution for intrusion detection systems. In *Intelligent Data Engineering and Automated Learning–IDEAL 2014: 15th International Conference, Salamanca, Spain, September 10-12, 2014. Proceedings 15*, pages 255–265. Springer, 2014.

C. Callegari, S. Giordano, M. Pagano, and T. Pepe. Combining sketches and wavelet analysis for multi time-scale network anomaly detection. *Computers & Security*, 30(8):692–704, 2011.

A. A. Cárdenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM symposium on information, computer and communications security*, pages 355–366, 2011.

A. Cherepanov and R. Lipovsky. Blackenergy–what we really know about the notorious cyber attacks. *Virus Bulletin October*, 2016.

CVE-2008-1084. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-1084`.

CVE-2008-3431. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-3431`.

CVE-2008-4250. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2008-4250`.

CVE-2010-2568. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568`.

CVE-2010-2729. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2729`.

CVE-2010-2772. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2772`.

CVE-2012-0158. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-0158`.

CVE-2013-3906. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-3906`.

CVE-2014-4114. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-4114`.

CVE-2021-22681. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-22681`.

CVE-2021-40449. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-40449`.

CVE-2022-21882. URL `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-21882`.

CVE-2022-21999. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-21999.

cyborgsec3. Hunting for persistence: Registry run keys / startup folder, Apr 2022. URL https://www.cyborgsecurity.com/cyborg-labs/hunting-for-persistence-registry-run-keys-startup-folder/.

J. P. S. da Silva and E. N. Cunha. Supervisory system for hidraulic distribution apparatus scadabr application. In *2017 IEEE First Summer School on Smart Cities (S3C)*, pages 161–164. IEEE, 2017.

Danielbohannon. Danielbohannon/invoke-obfuscation: Powershell obfuscator. URL https://github.com/danielbohannon/Invoke-Obfuscation.

M. V. De Assis, A. H. Hamamoto, T. Abrão, and M. L. Proença. A game theoretical based system using holt-winters and genetic algorithm with fuzzy logic for dos/ddos mitigation on sdn networks. *IEEE Access*, 5:9485–9496, 2017.

A. Dehlaghi Ghadima, A. Balador, H. Hanssona, and M. Contic. Icssim-a framework for building industrial control systems security simulation testbeds. *arXiv e-prints*, pages arXiv–2210, 2022.

S. Elsayed, R. Sarker, and J. Slay. Evaluating the performance of a differential evolution algorithm in anomaly detection. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2490–2497. IEEE, 2015.

J. M. Estevez-Tapiador, P. Garcia-Teodoro, and J. E. Diaz-Verdejo. Stochastic protocol modeling for anomaly based network intrusion detection. In *First IEEE International Workshop on Information Assurance, 2003. IWIAS 2003. Proceedings.*, pages 3–12. IEEE, 2003.

N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier. *White paper, symantec corp., security response*, 5(6):29, 2011.

J. P. Farwell and R. Rohozinski. Stuxnet and the future of cyber war. *Survival*, 53(1):23–40, 2011. doi: 10.1080/00396338.2011.555586. URL https://doi.org/10.1080/00396338.2011.555586.

G. Fernandes, J. J. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70:447–489, 2019.

D. Formby, M. Rad, and R. Beyah. Lowering the barriers to industrial control system security with grfics. In *2018 USENIX Workshop on Advances in Security Education (ASE 18)*, 2018.

B. N. Getu. Modelling and analysis of a nonlinear system using simulink. In *2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, pages 1–4. IEEE, 2019.

A. Ghaleb, S. Zhioua, and A. Almulhem. On plc network security. *International Journal of Critical Infrastructure Protection*, 22:62–69, 2018.

M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. *KI-2012: poster and demo track*, 1:59–63, 2012.

N. Govil, A. Agrawal, and N. O. Tippenhauer. On ladder logic bombs in industrial control systems. In *Computer Security: ESORICS 2017 International Workshops, CyberICPS 2017 and SECPRE 2017, Oslo, Norway, September 14-15, 2017, Revised Selected Papers 3*, pages 110–126. Springer, 2018.

P. Goyal and A. Goyal. Comparative study of two most popular packet sniffing tools-tcpdump and wireshark. In *2017 9th International Conference on Computational Intelligence and Communication Networks (CICN)*, pages 77–81. IEEE, 2017.

F. Gu, Q. Guo, L. Li, Z. Peng, W. Lin, X. Yang, and X. Gong. Comrace: Detecting data race vulnerabilities in com objects. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3019–3036, 2022.

A. H. Hamamoto, L. F. Carvalho, L. D. H. Sampaio, T. Abrão, and M. L. Proença Jr. Network anomaly detection system using genetic algorithm and fuzzy logic. *Expert Systems with Applications*, 92:390–402, 2018.

M. Hamdi and N. Boudriga. Detecting denial-of-service attacks using the wavelet transform. *Computer Communications*, 30(16):3203–3213, 2007.

C. Hammerschmidt, S. Marchal, R. State, G. Pellegrino, and S. Verwer. Efficient learning of communication profiles from ip flow records. In *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, pages 559–562. IEEE, 2016.

D. He, S. Chan, X. Ni, and M. Guizani. Software-defined-networking-enabled traffic anomaly detection and mitigation. *IEEE Internet of Things Journal*, 4(6):1890–1898, 2017.

Z. He, X. Xu, and S. Deng. Discovering cluster-based local outliers. *Pattern recognition letters*, 24 (9-10):1641–1650, 2003.

H. Holm, M. Karresand, A. Vidström, and E. Westring. A survey of industrial control system testbeds. In *Secure IT Systems: 20th Nordic Conference, NordSec 2015, Stockholm, Sweden, October 19–21, 2015, Proceedings*, pages 11–26. Springer, 2015.

T. Huang, H. Sethu, and N. Kandasamy. A new approach to dimensionality reduction for anomaly detection in data traffic. *IEEE Transactions on Network and Service Management*, 13(3): 651–665, 2016.

M. Huc. How to enable remote desktop from powershell on windows 10, Feb 2023. URL `https://pureinfotech.com/enable-remote-desktop-powershell-windows-10/`.

E. Kabir, J. Hu, H. Wang, and G. Zhuo. A novel statistical technique for intrusion detection systems. *Future Generation Computer Systems*, 79:303–318, 2018.

A. Karami and M. Guerrero-Zapata. A fuzzy anomaly detection system based on hybrid pso-kmeans algorithm in content-centric networks. *Neurocomputing*, 149:1253–1269, 2015.

G. S. Kc, A. D. Keromytis, and V. Prevelakis. Countering code-injection attacks with instruction-set randomization. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 272–280, 2003.

A. Keliris, H. Salehghaffari, B. Cairl, P. Krishnamurthy, M. Maniatakos, and F. Khorrami. Machine learning-based defense against process-aware attacks on industrial control systems. In *2016 IEEE International Test Conference (ITC)*, pages 1–10. IEEE, 2016.

D. Kennedy, J. O'gorman, D. Kearns, and M. Aharoni. *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

R. Khan, P. Maynard, K. McLaughlin, D. Laverty, and S. Sezer. Threat analysis of blackenergy malware for synchrophasor based real-time control and monitoring in smart grid. In *4th International Symposium for ICS & SCADA Cyber Security Research 2016 4*, pages 53–63, 2016.

S. Kumar et al. An emerging threat fileless malware: a survey and research challenges. *Cybersecurity*, 3(1):1–12, 2020.

A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. *ACM SIGCOMM computer communication review*, 34(4):219–230, 2004.

G. Li and Y. Wang. Differential kullback-leibler divergence based anomaly detection scheme in sensor networks. In *2012 IEEE 12th international conference on computer and information technology*, pages 966–970. IEEE, 2012.

F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 eighth ieee international conference on data mining*, pages 413–422. IEEE, 2008.

O. Lyak. Spoolfool: Windows print spooler privilege escalation (cve-2022-21999), Feb 2022.

G. F. Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning*. Insecure, 2009.

V. Mata. Windows print spooler vulnerability: Accenture, Jan 2021.

A. P. Mathur and N. O. Tippenhauer. Swat: A water treatment testbed for research and training on ics security. In *2016 international workshop on cyber-physical systems for smart water networks (CySWater)*, pages 31–36. IEEE, 2016.

A. Matrosov, E. Rodionov, D. Harley, and J. Malcho. Stuxnet under the microscope. *ESET LLC (September 2010)*, 6, 2010.

J. Mazel, P. Casas, Y. Labit, and P. Owezarski. Sub-space clustering, inter-clustering results association & anomaly correlation for unsupervised network anomaly detection. In *2011 7th international conference on network and service management*, pages 1–8. IEEE, 2011.

E. E. Miciolino, G. Bernieri, F. Pascucci, and R. Setola. Communications network analysis in a scada system testbed under cyber-attacks. In *2015 23rd Telecommunications Forum Telfor (TELFOR)*, pages 341–344. IEEE, 2015.

S. A. Milinković and L. R. Lazić. Industrial plc security issues. In *2012 20th Telecommunications Forum (TELFOR)*, pages 1536–1539. IEEE, 2012.

ModbusOrganization. *MODBUS Application Protocol Specification: V1. 1b3*. Modbus Organization, 2012.

T. H. Morris and W. Gao. Industrial control system cyber attacks. In *1st International Symposium for ICS & SCADA Cyber Security Research 2013 (ICS-CSR 2013) 1*, pages 22–29, 2013.

T. H. Morris, Z. Thornton, and I. Turnipseed. Industrial control system simulation and data logging for intrusion detection system research. *7th annual southeastern cyber security summit*, pages 3–4, 2015.

Y. Movahedi, M. Cukier, A. Andongabo, and I. Gashi. Cluster-based vulnerability assessment of operating systems and web browsers. *Computing*, 101:139–160, 2019.

C. Nachreiner. Anatomy of an arp poisoning attack. *Retrieved July*, 4:2005, 2003.

C. O'Reilly, A. Gluhak, and M. A. Imran. Distributed anomaly detection using minimum volume elliptical principal component analysis. *IEEE Transactions on Knowledge and Data Engineering*, 28(9):2320–2333, 2016.

B. Ortiz. Creating av resistant malware-part 1, Dec 2020a. URL `https://blog.securityevaluators.com/creating-av-resistant-malware-part-1-7604b83ea0c0?gi=74d4eb2e2b89`.

B. Ortiz. Creating av resistant malware-part 2, Dec 2020b. URL https://blog.securityevaluators.com/creating-av-resistant-malware-part-2-1ba1784064bc.

PacketStormSecurity. Win32k ntgdiresetdc use-after-free / local privilege escalation, a. URL https://packetstormsecurity.com/files/164926/Win32k-NtGdiResetDC-Use-After-Free-Local-Privilege-Escalation.html.

PacketStormSecurity. Win32k consolecontrol offset confusion / privilege escalation, b. URL https://packetstormsecurity.com/files/166169/Win32k-ConsoleControl-Offset-Confusion-Privilege-Escalation.html.

PacketStormSecurity. Windows spoolfool privilege escalation, c. URL https://packetstormsecurity.com/files/166344/Windows-SpoolFool-Privilege-Escalation.html.

S. Rahalkar and N. Jaswal. Metasploit revealed: Secrets of the expert pentester. URL https://www.oreilly.com/library/view/metasploit-revealed-secrets/9781788624596/e1d79ad3-0ba0-41e4-9082-9590cfa0089e.xhtml.

A. Rahman, G. Mustafa, A. Q. Khan, M. Abid, and M. H. Durad. Launch of denial of service attacks on the modbus/tcp protocol and development of its protection mechanisms. *International Journal of Critical Infrastructure Protection*, 39:100568, 2022.

Raiu. Mysterysnail attacks with windows zero-day, May 2021. URL https://securelist.com/mysterysnail-attacks-with-windows-zero-day/104509/.

S. Rajasegarar, C. Leckie, and M. Palaniswami. Hyperspherical cluster based distributed anomaly detection in wireless sensor networks. *Journal of Parallel and Distributed Computing*, 74(1): 1833–1847, 2014.

L. Rosa, T. Cruz, P. Simões, E. Monteiro, and L. Lev. Attacking scada systems: A practical perspective. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 741–746. IEEE, 2017.

G. Sandaruwan, P. Ranaweera, and V. A. Oleshchuk. Plc security and critical infrastructure protection. In *2013 IEEE 8th international conference on industrial and information systems*, pages 81–85. IEEE, 2013.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural computation*, 13(7):1443–1471, 2001.

Secdev. Secdev/scapy: Scapy: The python-based interactive packet manipulation program & library. supports python 2 & python 3., 2022. URL https://github.com/secdev/scapy.

A. Serhane, M. Raad, R. Raad, and W. Susilo. Plc code-level vulnerabilities. In *2018 International Conference on Computer and Applications (ICCA)*, pages 348–352. IEEE, 2018.

S. Shamshirband, N. B. Anuar, M. L. M. Kiah, V. A. Rohani, D. Petković, S. Misra, and A. N. Khan. Co-fais: cooperative fuzzy artificial immune system for detecting intrusion in wireless sensor networks. *Journal of Network and Computer Applications*, 42:102–117, 2014.

C. E. Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3): 379–423, 1948.

A. Sharma, B. B. Gupta, A. K. Singh, and V. Saraswat. Orchestration of apt malware evasive manoeuvers employed for eluding anti-virus and sandbox defense. *Computers & Security*, 115: 102627, 2022.

L. Shi, S. Krishnan, and S. Wen. Study cybersecurity of cyber physical system in the virtual environment: A survey and new direction. In *Australasian Computer Science Week 2022*, pages 46–55. 2022.

M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, Miami Univ Coral Gables Fl Dept of Electrical and Computer Engineering, 2003.

M. P. Silva, A. L. Gonçalves, M. A. Dantas, B. Vanelli, G. Manerichi, S. A. Dos Santos, M. Ferrandim, and A. R. Pinto. Implementation of iot for monitoring ambient air in ubiquitous aal environments. In *2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 158–161. IEEE, 2015.

J.-H. Song, Y. Her, J. Park, K.-D. Lee, and M.-S. Kang. Simulink implementation of a hydrologic model: a tank model case study. *Water*, 9(9):639, 2017.

M. G. Stefan Nicula. Anatomy of an exploit in windows win32k - cve-2022-21882, Aug 2022. URL `https://www.avira.com/en/blog/anatomy-of-an-exploit-in-windows-win32k-cve-2022-21882`.

Stephenfewer. Stephenfewer/reflectivedllinjection. URL `https://github.com/stephenfewer/ReflectiveDLLInjection`.

J. Stewart. Blackenergy version 2 threat analysis. *SecureWorks. Haettu*, 29:2021, 2010.

B. Strom, B. Strom, A. Applebaum, D. Miller, K. Nickels, A. Pennington, and C. Thomas. Mitre att&ck: Design and philosophy, Mar 2020. URL `https://www.mitre.org/news-insights/publication/mitre-attck-design-and-philosophy`.

M.-Y. Su. Discovery and prevention of attack episodes by frequent episodes mining and finite state machines. *Journal of Network and Computer Applications*, 33(2):156–167, 2010.

M. Swarnkar and N. Hubballi. Ocpad: One class naive bayes classifier for payload based anomaly detection. *Expert Systems with Applications*, 64:330–339, 2016.

D. M. Tax and R. P. Duin. Support vector data description. *Machine learning*, 54:45–66, 2004.

M. Tiegelkamp and K.-H. John. *IEC 61131-3: Programming industrial automation systems*, volume 166. Springer, 2010.

M. Tigner, H. Wimmer, and C. M. Rebman. Windows reverse tcp attack: The threat of out-of-date machinery. In *2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 555–560. IEEE, 2021.

Trellix. What is the mitre att&ck framework?: Get the 101 guide. URL `https://www.trellix.com/en-us/security-awareness/cybersecurity/what-is-mitre-attack-framework.html`.

G. Tzokatziou, L. A. Maglaras, H. Janicke, and Y. He. Exploiting scada vulnerabilities using a human interface device. *Int J Adv Comput Sci Appl*, pages 234–241, 2015.

H. Wardak, S. Zhioua, and A. Almulhem. Plc access control: a security analysis. In *2016 World Congress on Industrial Control Systems Security (WCICSS)*, pages 1–6. IEEE, 2016.

S. Wen, W. Zhou, J. Zhang, Y. Xiang, W. Zhou, W. Jia, and C. C. Zou. Modeling and analysis on the propagation dynamics of modern email malware. *IEEE transactions on dependable and secure computing*, 11(4):361–374, 2013.

L. Williams. Wireshark tutorial: Network & passwords sniffer, Feb 2023. URL `https://www.guru99.com/wireshark-passwords-sniffer.html`.

T. J. Williams. The purdue enterprise reference architecture. *Computers in Industry*, 24(2):141–158, 1994. ISSN 0166-3615. doi: https://doi.org/10.1016/0166-3615(94)90017-5. URL `https://www.sciencedirect.com/science/article/pii/0166361594900175`.

M. Xie, J. Hu, and S. Guo. Segment-based anomaly detection with approximated sample covariance matrix in wireless sensor networks. *IEEE Transactions on Parallel and Distributed Systems*, 26(2):574–583, 2014.

M. Xie, J. Hu, S. Guo, and A. Y. Zomaya. Distributed segment-based anomaly detection with kullback–leibler divergence in wireless sensor networks. *IEEE Transactions on Information Forensics and Security*, 12(1):101–110, 2016.

Y. Xie, W. Wang, F. Wang, and R. Chang. Vtet: A virtual industrial control system testbed for cyber security research. In *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–7. IEEE, 2018.

T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 5th International Conference, DIMVA 2008, Paris, France, July 10-11, 2008. Proceedings 5*, pages 207–227. Springer, 2008.

T. Yigit and O. F. Selamet. Mathematical modeling and dynamic simulink simulation of high-pressure pem electrolyzer system. *International Journal of Hydrogen Energy*, 41(32):13901–13914, 2016.

I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *2010 International conference on broadband, wireless computing, communication and applications*, pages 297–300. IEEE, 2010.