UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

EXPLORING SECURITY CONTROLS FOR ICS/SCADA ENVIRONMENTS

Pedro Miguel França Barreiros

MESTRADO EM SEGURANÇA INFORMÁTICA

Trabalho de projeto orientado por:
Prof. Doutor Bernardo Ferreira

2020

Acknowledgments

I begin by thanking my internal co-advisor at Siemens Portugal, Engineer Ricardo Carona, team leader of *ProductCERT*, and my advisor in the IT department at the Faculty of Sciences of the University of Lisbon, Professor Bernardo Ferreira for the follow-up, support and patience that they managed to give me over these several months. To my co-advisor Ricardo Carona I also thank you for the knowledge you have provided and the help given to overcome the ”blocking” moments of my critical and development process.

I am also grateful to the other colleagues in the team where I was inserted (*ProductCERT*) who always pushed for me and gave me academic, technical and fraternal support. I especially name Julio Morais, Pedro Brandão, João Lopes and Gonçalo Louro. Still within the *ProductCERT* team, I would also like to thank my colleague Marco Vieira, master's student, for the daily company and help in the development of our dissertations. Special thanks also to Sebastião Nunes and Patricia Mauricio, from the human resources team, for keeping me under their eyes and helping me with any type of existential question.

I end by thanking all my friends and family who have given me immense support in my personal life and helped me to maintain my sanity throughout this journey. A big, big thank you to Beatriz Sécio, Diogo Soares, Daniel Ramos and Diogo Ferreira for being a great foundation and great friends with whom I have always been able to count on, both for academic and personal issues. To all the friends from my small town, always available to hear my outburst on Friday nights, João Pires, Alexandre Domingos, Mariana Freire, Eduardo Brito, Miguel Condeça, João Araujo and many others, a huge hug. Emanuel Eugenio, my longtime brother, thanks for everything. And to my parents, thank you very much for being who you are and for instilling respect and providing me with the tools to reach the end of this stage. I love you.

To everyone who accompanied me throughout my academic journey.

Resumo

Os Sistemas de Controlo Industriais (ICS) estão a começar a fundir-se com as soluções de IT, por forma a promover a interconectividade. Embora isto traga inumeros beneficios de uma perspetiva de controlo, os ICS apresentam uma falta de mecanismos de segurança que consigam evitar possíveis ameaças informáticas, quando comparados aos comuns sistemas de informação [29], [64]. Dada a natureza crítica destes sistemas, e a ocorrências recentes de ciberataques desastrosos, a segurança é um tópico que deve ser incentivado. À luz deste problema, na presente dissertação apresentamos uma avaliação de possíveis aplicações e controlos de segurança a serem implantados nestes ambientes críticos e a implementação de uma solução de segurança extensível que dá resposta a certos ataques focados em sistemas industriais, capaz de ser implantada em qualquer rede industrial que permita a sua ligação. Com o auxilio de uma *framework* extensivel e portatil para testes de ICS, e outros ambientes industriais de testes, foi possível analisar diferentes cenários de ameaças, implantar mecanismos de segurança para os detetar e avaliar os resultados, com o intuito de fornecer uma ideia de como empregar estes mecanismos da melhor maneira possível num ambiente real de controlo industrial.

Palavras-chave: ICS, IDS, Redes, Segurança

Abstract

Industrial Control Systems (ICS) are beginning to merge with IT solutions, in order to promote inter-connectivity. Although this brings countless benefits from a control perspective, ICS have been lacking in security mechanisms to ward off potential cyber threats, when compared to common information systems [29], [64]. Given the critical nature of these systems, and the recent occurrences of disastrous cyber-attacks, security is a topic that should be encouraged. In light of this problem, in this dissertation we present an assessment of possible security applications and controls that can be deployed in these critical environments and the implementation of an extensible security solution that responds to certain attacks focused on industrial systems, capable of being deployed in any industrial network that allows its connection. With the help of an extensible and portable *framework* for ICS testing, and other industrial testing environments, it was possible to analyze different threat scenarios, implement security mechanisms to detect them and evaluate the results in order to provide an idea on how to employ these mechanisms as best as possible in a real industrial control environment, without compromising it's process.

Keywords: ICS, IDS, Network, Security

Contents

List of Figures

List of Tables

Acronyms

APT	Advanced Persistent Threat
ARP	*Address Resolution Protocol*
CPU	*Central Processor Unit*
CERT NetSA	*CERT Network Situational Awareness Team*
COTP	*Connection Oriented Transport Protocol*
DHCP	*Dynamic Host Configuration Protocol*
DMZ	*Demilitarized Zone*
DNP3	*Distributed Network Protocol 3*
DNS	*Domain Name System*
DoS	*Denial of Service*
ELK Stack	*Elastic Stack*
HMI	*Human Machine Interface*
HIDS	*Host-based Intrusion Detection System*
ICS	*Industrial Control System*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IED	*Intelligent Electronic Device*
IPS	*Intrusion Prevention System*
IT	*Information Technologies*
MLFB	*Maschinen lesbare Fabrikate bezeichnung*
MMS	*Manufacturing Message Specification*
MTIM	*Man-in-the-Middle*
NIDS	*Network-based Intrusion Detection System*
NIST	*National Institute of Standards and Technology*
OT	*Operational Technologies*

PLC	*Programmable Logic Controller*
RTU	*Remote Terminal Unit*
S7Comm	*S7 Communication*
SCADA	*Supervisory Control and Data Acquisition*
SIEM	*Security information and event management*
SiLK	System for Internet-Level Knowledge
TCP	*Transmission Control Protocol*
TLV	*tag-length-value*
TPKT	*ISO Transport Service on top of the TCP*
TTP	*Techniques Tactics and Procedures*
UDP	*User Datagram Protocol*
VMD	*Virtual Manufactoring Device*
VPN	*Virtual Private Network*

Chapter 1

Introduction

Critical infrastructures and industrial ecosystems, such as nuclear plants; transport systems; power grids; water treatment plants; oil and gas distribution systems and overall production systems are becoming increasingly reliant on machines and automated processes as technology evolves. As such, it is imperative that these machines be reliable, robust, and safe. *Industrial Control System* (ICS) and *Supervisory Control and Data Acquisition* (SCADA) systems, which are responsible for monitoring and controlling industrial processes and operations, have been designed to accommodate these requirements as best as possible, and yet, system security remains to be touched upon [29], [64].

Cyber-attacks have been a huge problem for ICS/SCADA systems, which can result in catastrophic outcomes from lack of security features and monitoring. Some recent occurrences of such attacks, like the Stuxnet malware [18], in 2010, and BlackEnergy malware [37], in 2015, among others, can be seen as two examples of cyber-attacks coming to fruition from lack of security mechanisms. Stuxnet was a malware designed to physically sabotage the gas centrifuges in Iran's uranium enrichment plant, all the while, providing "expected" information to the monitoring system, to not raise suspicion, resulting in centrifuge damage. The BlackEnergy malware, leveraged VPN to infiltrate the Ukrainian power grid control center and, consequently, tamper with the relay's control instructions to cut-off power circulation. Although the resulting damage and loss of production, in the previously mentioned cases, was terrible, even more disastrous outcomes could come to fruition because of these disruptions – inducing a negative impact on public safety and further economic losses.

It is with this in mind that detecting possible attacks in a SCADA/ICS network, as to avoid the compromise of further devices or even the infrastructure itself, becomes an overbearing issue. *Intrusion Detection System* (IDS), which, as the name suggests, help detect attacks that may be taking place on the network and/or on its devices, become a very important asset in ICS. Although IDSs are a common security feature for most IT systems, the same cannot be said for Industrial

Control Systems [5]. Security features have been lacking in the ICS domain, as it was never a necessity, given the fact that the control network in critical infrastructures was usually separated from the enterprise network – "air-gap" [29]. As soon as some quality-of-life improvements, like remote monitoring and control, were implemented, new attack surfaces were introduced in these critical infrastructures.

In light of this issue, the main objective of this work, is to understand what possible security applications could make sense in an ICS context by implementing an IDS solution and other relevant security mechanisms on an extensible testing framework that simulates an ICS/SCADA, and other ICS testing environments, in order to detect potential attacks on those networks, while also providing enough information to make an assessment on what better tools to use, which vantage points provide the best detection surface and also what other real world use-case scenarios could be feasibly deployed, if resources , especially time-wise, were made available.

1.1 Motivation

A great amount of research has already been conducted regarding network protection for IT systems. Although this is true for a normal IT setting, the same cannot be said for ICS/SCADA [5]. These systems have been lacking in defense mechanisms, which becomes even more important as these systems start to promote inter-connectivity. Because of this, research on this topic has seen an increase in recent years [26].

In an ICS/SCADA setting, security mechanisms must refrain from halting production, requiring non invasive solutions like an IDS solution. However, since this is a relatively recent topic, IDS technologies haven't seen much contextualization with ICS requirements.

Industrial control systems must be allowed to flourish into a fully automated and/or remote-controlled process for critical infrastructures without the fear of compromise by the growing technological advances in Cyber-attacks. And so, security mechanisms must be employed to deal with these intrusions.

1.2 Objective

The focus of this dissertation is to help improve ICS defense topics by becoming familiar with the ICS world, learning how to obtain relevant security logs from all devices, network traffic and act upon this information in order to understand what security mechanisms could be deployed on a security testing framework for ICS and, ultimately, implement a security appliance capable of ensuring detection and monitoring for ICS-based intrusions. In order to do so, it is imperative to understand ICS TCP based protocols; resort to scripting to allow for parsing and filtering of secu-

rity logs; and the creation and manipulation of IDS rules and detection policies for an integrated intrusion detection and monitoring solution.

The goal is to create a secure perimeter around an extensible testing framework based on ICS/SCADA with real devices, by deploying and researching several security mechanisms (e.g. IDS solutions, SIEM solutions, Firewall solutions, etc.), make an assessment on their effectiveness at detecting and/or preventing intrusions and settling on the implementation of a fully integrated intrusion detection and monitoring solution.

1.3 Contributions

The main contributions of this work are:

- An analysis of different security mechanisms that could be deployed on these environments was conducted to help decide on a specific solution to implement;
- After analyzing potential mechanisms, an ICS security appliance was implemented. This appliance utilizes both Snort-IDS and Zeek-IDS to detect intrusions happening on the network, based on ICS threats, along with the Bropy tool for baselining traffic. All the intrusion and log information is configured to be shipped to a SIEM solution that helps grant visibility to the monitoring environment.
- The solution implemented is easily and highly extendable, to improve defense from threats specific to the exact environment the appliance will be deployed on. It ensures detection on some ICS attack examples while also establishing a framework for further development and support.

1.4 Document Structure

This document's structure is as follows:

1 Introduction: An introduction to the general topics and notions required to understand and contextualize the scope of this thesis. An overview of ICS/SCADA systems and the problem statement.

2 Background and Related Work: This chapter, encompasses an overview of technologies to be approached and delved into in this work, as well as, providing a summary of investigation and works regarding the overall goal of the thesis.

3 Analysis of industrial Attacks and security mechanisms: An overview of security mechanisms and tests conducted to decide on a fully integrated security solution, along with example of an ICS attack pathway, will be discussed.

4 ICS-Security-Appliance This chapter is the focal point of this thesis. It depicts all the implementation and ideas undergone to develop an integrated intrusion detection and monitoring solution. The ICS-Security-Appliance. It details how baselining traffic is done, some detection use-cases and how they were implemented using Zeek and/or Snort, as well as how Elastic Stack (SIEM) is configured.

5 Evaluating the ICS Security Appliance In this chapter, the topics discussed are of an evaluative nature. Every aspect pertaining to the ICS-Security-Appliance implementation will be evaluated here. An evaluation on the baselining mechanisms; an analysis of the importance of the configured SIEM solution, along with an assessment of its use-cases; some important aspects with regards to the implemented detection use-cases (which worked and which didn't and why) and, finally, an assessment on the strengths and short-comings of the ICS-Security-Appliance itself.

6 Conclusions and Future Work The final chapter relates to closing thoughts and possibilities for future work.

Chapter 2

Context and related work

The fundamentals and overview of technologies for a general understanding of this dissertation are discussed below. I start by explaining what Industrial Control Systems are, then delve slightly deeper into a subset of these systems, enlighten on some ICS network architecture notions, introduce the communication protocols present in ICS environments, explain what Intrusion Detection Systems are and their possible applications, SIEMs and other security applications. Some related work that makes sense in the scope of this dissertation will also be presented, as well as other fundamental security aspects to grant further context.

2.1 The ICS setting

2.1.1 ICS Overview

Industrial control systems

Industrial Control Systems (ICS) are systems allied with critical infrastructures and industrial ecosystems, responsible for the industrial process and control of operations. They make use of several components like Programmable Logic Controllers (PLCs), Human Machine Interfaces (HMIs), sensors and actuators in order to control and monitor the production of said critical infrastructure. These systems receive data from remote sensors, responsible for measuring process variables from field devices and components, comparing them with desired variables (set points) and relay commands to the components to update instructions or poll specific information [64].

Supervisory Control and Data Acquisition systems

Supervisory Control And Data Acquisition (SCADA) is a subset of ICS that provides centralized monitoring and control, which handles a large amount of inputs and outputs. The information

collected from field devices and components is transferred to a central computer facility in order to display that information to operators [64].

ICS Components

Introducing some ICS specific components to add further context into industrial environments.

- **PLC** - *Programmable Logic Controllers* are the workhorse of industrial automation. They are special purpose computers that contain a CPU module that controls I/O devices (which can be stuck to the PLC or connected through data cables) through logic programs implemented on the CPU.

- **HMI** - *Human Machine Interface* allow operators to control and monitor machines through a visual display. An HMI portrays information like temperature, pressure, process phases, liquid levels in tanks, etc. The HMI connects to a PLC through Inputs and Outputs.

- **Field I/O Devices** - *Field Input/Output Devices* are components, sensors and actuators that remain in the process area of an industrial environment, responsible for receiving control instructions from control devices (i.e. PLCs) and collecting process data (pressure; temperature; etc.). and relay them to an HMI.

- **RTU** - *Remote Terminal Unit* is a device based on microprocessors that monitors and controls field devices. Connects to the industrial control plant and/or to SCADA. RTUs functions much like a PLC but have greater environment tolerance and greater remote coverage.

- **IED** - *Intelligent Electronic Devices* also establish a connection between remote sensors and actuators, much like the PLCs and RTUs.

- **Data Historian** - An ICS data historian is a time series database where industrial process and control information is archived. Analog and digital readings from sensors; previously sent control instructions; equipment state changes; etc. It is often used to correlate information and obtain visibility into the industrial environment [55];

- **Engineering Workstations** - It is from engineering workstations that plant developers implement the desired programmable logic to control devices.

To note, Siemens PLCs (IEDs and RTUs also) and HMIs are represented by an identification code called MLFB (Maschinen lesbare Fabrikate bezeichnung - Machine-Readable Product Designation). This will be discussed later in Chapter 4.

2.1.2 IT vs OT

The scope of this dissertation lies within the Industrial setting, so it is important to distinguish Operational Technologies (OT) from Information Technologies (IT), in order to give a fundamental understanding of the different underlying security necessities and requirements that make sense in this circumstance.

Operational Technologies (OT) refer to hardware and software systems responsible for monitoring and controlling industrial processes, as seen in ICS. Performance and availability requirements are much more constrained; architecture security focuses on guaranteeing the availability of field devices; and risk management falls under human safety and fault tolerance notions. In IT, Architecture security focuses on IT assets and the main risk management concern is data confidentiality and integrity [64].

ICSs are deployed in isolated infrastructures with their own proprietary communication protocols, which are different from traditional IT environments [27]. Historically, the only way to access the system nefariously was by gaining physical access to the facility. System security was reliant on "security by obscurity" and lack of external connectivity, the "air-gap" [6]. Such measures are no longer enough to protect these systems, as OT are adopting IT solutions to monitor and control industrial process remotely and integrate it with enterprise connectivity [64],[30], [29]. "OT cyber security is roughly a decade behind the maturity level of IT security in many ways, including organizational development, funding, available tools and skilled resources." [27]. Security deployment in OT is an undeveloped thought but has been subject to change.

According to SANS [27], ICS/OT networks possess a static nature and harbor lower volumes of network traffic, when compared to normal IT environments. Traffic is a product of component communication, which are pre-configured given the production needs. There isn't much deviation from the expected communication, aside from the occasional update or reconfiguration from a control center or engineering workstation. If the network components and devices are well inventoried and their inter-communications well documented, it can be inferred that industrial activity is extremely deterministic. Being deterministic makes it easier to formulate a traffic and process baseline to help in the detection of possible attacks, with the help of an intrusion detection system (IDS) (explained further in Section 2.2). Table 2.1 depicts an overview of the differences between IT and OT settings.

Category	IT	OT
Architecture Security	Ensure security of IT assets	Ensure availability of field devices
Risk Management	Data confidentiality and Integrity	Human safety and Fault tolerance
Environment	Dynamic and High volume of traffic	Static, Deterministic and low volume of traffic

Table 2.1: IT vs OT

2.1.3 ICS Network Architecture

Segmentation is the backbone of ICS Network Architectures, helping to safeguard the network from potential intrusions [27], [10]. It is recommended that network segments referring to industrial control systems be kept separate from the enterprise-network – which has a greater need for internet and extranet connectivity [10], [64], [30], [29]. Thus, network segments should be separated according to their purpose and necessities. Typically, segmentation should divide the Industrial network in three layers: an Industrial Control Network; an Internal Demilitarized Zone (DMZ) and an Enterprise Area Network.

For this reason, the Purdue model was developed to be a framework to help form a network architecture for ICS centered environments [49]. Through the Purdue model, systems performing similar functions are divided into zones of similar requirements, denoted by levels.

- Level 05 - Enterprise Network
- Level 04 - Site business planning and logistics
- Level 03 - Site manufacturing operation and control
- Level 02 - Supervisory Control
- Level 01 - Basic control
- Level 00 - Process

Figure 2.1 represents an example of an architecture, similar to NIST's (National Institute of Standards and Technology) architecture recommendation [49], utilizing the Purdue model. At level 05, servers pertaining to the enterprise external DMZ, such as VPN and Web servers, as well as some other corporate servers for accounting and business applications, are located; Level 04 entails the location of corporate servers that include E-mail, Print and inventory services, in addition, IT services like active directory, DNS, DHCP are also present, as well as employee workstations; A Plant DMZ, responsible for separating the enterprise network from the industrial control network, sits between level 04 and level 03 with remote access servers for accessing the industrial control network via the enterprise network, as well as a replicated historian server, storing industrial information. It is of utmost importance to separate both the enterprise network from the industrial control network.

An industrial control network can usually be comprised of four segments, *Site manufacturing operation and control* at level 03, where systems responsible for managing control plant operations are located (i.e Plant Historian, SCADA server, Engineering Workstations, Remote access services, etc), *Supervisory Control* at level 02, where monitoring and control devices are placed

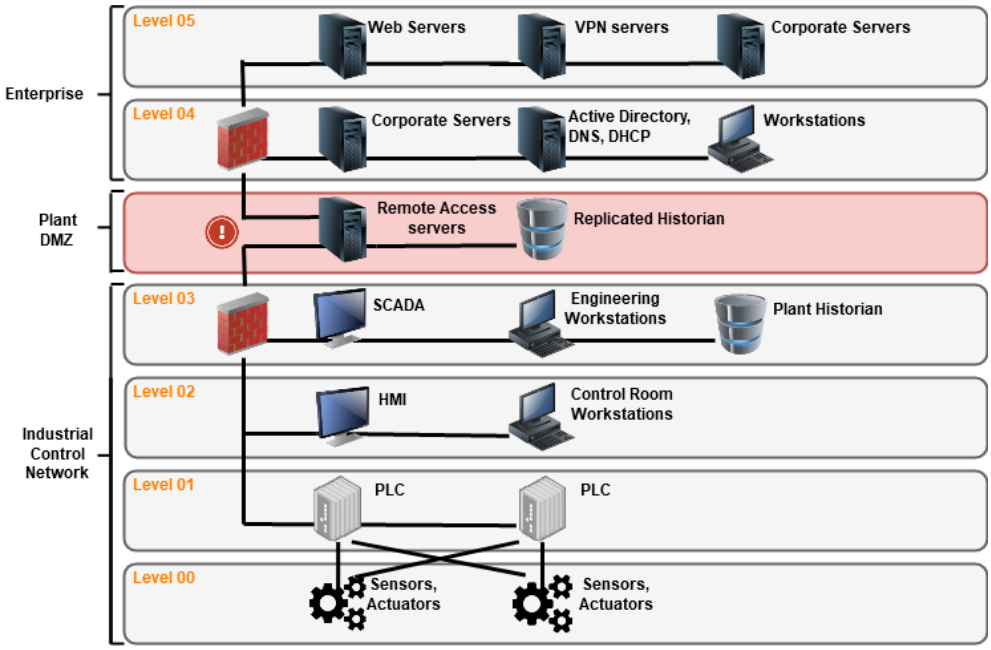

Figure 2.1: Example of architecture based on NIST recommendation and the purdue model

(i.e HMIs, Control Room Workstations and Control Servers); *Basic Control* at level 01, where the main control components and intelligent devices are located (i.e PLCs, RTUs and IEDs); and *Physical Process* at level 00, where field I/O devices are operating (i.e. sensors, actuators, valves, etc.).

2.1.4 Communication Protocols

This section gives an insight into communication protocols in Industrial Control Systems and their security features, or lack there of. selecting the protocols that best represent the tests to be conducted as this work progresses, based on different industry architectures.

Communication between the different network layers should employ encryption procedures. Although this is a very important security measure, most ICS communication protocols do not incorporate encryption methods in their operation [10]. As will be discussed further in this section.

Protocol	Description	Security
Modbus	Master/Slave communication via TCP	No Authentication; No Encryption; No Validation; No Broadcast Suppression
DNP3	Master/Slave communication via TCP and UDP	No Authentication; No Encryption
S7 Comm	Master/Slave communication via TCP	No integrity protection; No confidentiality protection; Message injection and modification is possible
IEC 61850	Client/server communication via TCP	Access control list; Vulnerable to replay attacks

Table 2.2: Industrial Control Protocols

Modbus

Modbus was initially implemented for serial communication, but for the sake of this dissertation, I will limit the explanation to the later developed version, Modbus over TCP.

Modbus is a widely-accepted, simple and lightweight, protocol for industrial communication, with a data limit of 253 bytes. It uses a ”request/reply” model between interconnected devices and encompasses a Master/Slave (client/server) architecture where only one device can initiate queries. [51]

The slaves/server supply the requested data to the master/client or perform the action requested by the master/client. A slave can be any device (valve, measuring devices, sensors, actuators) which processes information and relays it to the master. [51]

Modbus communicates by Function Codes. A Function code can be used to perform a wide range of commands, which can be seen in table 2.3

Function Code	**Function name**
0x01	Read Coil Status
0x02	Read Input Status
0x03	Read Holding Registers
0x04	Read Input Registers
0x05	Force Single Coil
0x06	Preset Single Register
0x07	Read Exception Satus
0x08	Diagnostics
0x09	Program 484
0x0A	Poll 484
0x0B	Fetch Communication Event Counter
0x0C	Fetch Communication Event Log
0x0D	Program Controller
0x0E	Poll Controller
0x0F	Force Multiple Coils
0x10	Preset Multiple Registers
0x11	Report Slave ID
0x12	Program 884/M84
0x13	Reset Communication Link
0x14	Read General Reference
0x15	Write General Reference
0x16	Mask Write 4X Register
0x17	Read/Write 4X Registers
0x18	Read FIFO Queue

Table 2.3: Modbus function codes

Within function code 08 (**Diagnostics**) Modbus has numerous sub-function codes. It is im-

portant to refer that sub-function code 04, entails a *Force Listen Only Mode* instruction [21]. This instruction can be used to provoke a denial of service condition (explained in 2.5.2) on some Modbus enabled systems (This will be discussed further in 4.8.13).

PLCs have memory which is used to store programs variables and information. In Modbus devices, all areas consist of memory blocks, called Coils, Inputs, Input Registers and Holding registers. These memory blocks influence different types data on PLC connections to the field devices they control. Discrete Inputs, which connect sensors like switches with an on an of button; Discrete Outputs, which connect actuators to open and close a valve, for example; Analog Inputs, which connect pressure and temperature sensors to get those readings from the PLC, for example; Analog outputs, responsible for allowing the PLCs to control the field device by managing variables that influence process. Table 2.4 represents the relationship between Modbus memory areas and the type of data associated with different PLC connections to controlled field devices.

Memory Area	Data	Example
Coils	Discrete Outputs	Actuators
Inputs	Discrete Inputs	Sensors
Input Registers	Analog Inputs	Pressure, Temperature, etc.
Holding Registers	Analog Outputs	Controls device

Table 2.4: Modbus memory area

Modbus Security

- Authentication - There is no authentication. A packet can be crafted with malicious data and sent to Modbus devices;
- Encryption - Modbus communication is in clear text;
- Validation - checksum is only validated on the transport layer;
- Broadcast Suppression - There is no broadcast suppression (all addresses receive all messages). Can lead to a Dos condition;
- Function Codes - The Simplicity of function codes allow for straight forward attacks-

DNP3

The DNP3 protocol , developed by Westronic is particularly used in oil and gas industries, as well as electric and water utilities. Can be encapsulated on TCP or UDP packets and typically utilizes port 2000 in its communications [50] [10].

For example, in Electric utilities, the usual setting for DNP3 protocol, starts with a control

center (normally a SCADA environment) and ends on a remote substation, responsible for transforming electric voltage, which roosts control devices and field I/O and components.

Normally, DNP3 is implemented in a Master-Slave configuration, just like Modbus. Where the control center (HMIs or SCADA) poses as the Master (Client), issuing commands and polling data, while every control device (RTU, PLC, IED, etc.) poses as a Slave (Server) responding to commands. Furthermore, DNP3 also works through function codes. Table 2.5 contains a list of some of the available function codes for DNP3.

Function Code	Function name
0x00	Confirm Function Code
0x01	Read Function Code
0x02	Write Function Code
0x03	Select Function Code
0x04	Operate Function Code
0x05	Direct Operate Function Code
0x0d	Cold Restart Function Code
0x0e	Warm Restart Function Code
0x12	Stop Application Function Code
0x14	Enable Unsolicited Messages Function Code
0x15	Disable Unsolicited Messages Function Code
0x1b	Delete File Function Code
0x81	Response Function Code
0x82	Unsolicited Response Function Code

Table 2.5: DNP3 function codes

It is important to note, from Table 2.5 there are two particular function codes of interest, 0x12 (Stop Application) and 0x15 (Disable Unsolicited Messages). Function code 0x12 when sent to a slave devices, stops the service from running which can cause a DoS condition on the device. Function code 0x15 when sent to a slave devices, stops status and error messages from being sent to the master control center, which can produce a loss in visibility. Both of these cases will be discussed further in Section 4.8.9 and Section 4.8.14 respectively.

DNP3 Security

- Authentication - There is no authentication. A packet can be crafted with malicious data and sent to DNP3 hosts;
- Encryption - Modbus communication is in clear text;
- Validation - checksum is validated on data link layer and the transport layer;
- Function Codes - The Simplicity of function codes and data types allow for straight forward attacks.

S7Comm

Gyorgy Miru does a great job in explaining the fundamentals of the S7Comm protocol in [43], and [44].

S7comm is utilized, for the most part, between a PLC and PC Stations (A control station with SIMATIC components - Engineering Workstation). S7Comm, a Siemens proprietary communication protocol, employs a traditional Master - Slave (Client - Server) model, much like DNP3 and Modbus. The PC Station (Master/Client) sends s7comm requests to the PLC (Slave/Server)

S7Comm is encapsulated in TPKT and ISO-COTP protocols to allow the s7 PDU (Protocol Data Unit) to be transferred via TCP/IP. Inside the s7comm payload, Jobs are requests sent from the Master to the Slave, which responds with Ack Data.

On Sections 4.8.7 (*program logic download*), 4.8.8 (*program logic upload*) and 4.8.9 (*service stop*) three command instructions, leveraging the s7comm to conduct an attack, will be presented. To that end, the remainder of this section will explain how these commands are organized within the s7comm PDU. A function code also dictates which instruction is being issued, much like in DNP3 and Modbus, some relevant examples are depicted in Table 2.6.

Function Code	Function name
0x1a	Request Download
0x1b	Download Block
0x1c	Download Ended
0x1d	Start Upload
0x1e	Upload Block
0x1f	End upload
0x29	PLC Stop

Table 2.6: S7Comm function codes

On Siemens devices, logic code and other data is stored in blocks. these blocks are described as follows:

- OB: The Organization Block stores the main program logic (The OB calls every other block necessary to run the program);
- DB: Data blocks stores every data required by the program running on the PLC;
- FC: These are stateless functions with some logic, used to be called from other programs;
- FB: Function Blocks are stateful functions, much like the OB, with their own associated DB for more complex logic.

In S7Comm, Downloading happens when a master sends data in blocks to the slave in order to

load program logic code to its hardware. An Upload happens when the Master pools data blocks from the slave to retrieve the program logic code running inside the slave. For this to happen a function code (detailed in Table 2.6), in the first byte of the Job payload (request), is sent to the slave, while the 10th byte describes the file name in question (the block we are trying to download or upload). For example, as Figure 2.2 entails, the first byte of the payload contains a request for downloading a block, while byte 10 describes it to be an OB1 block.

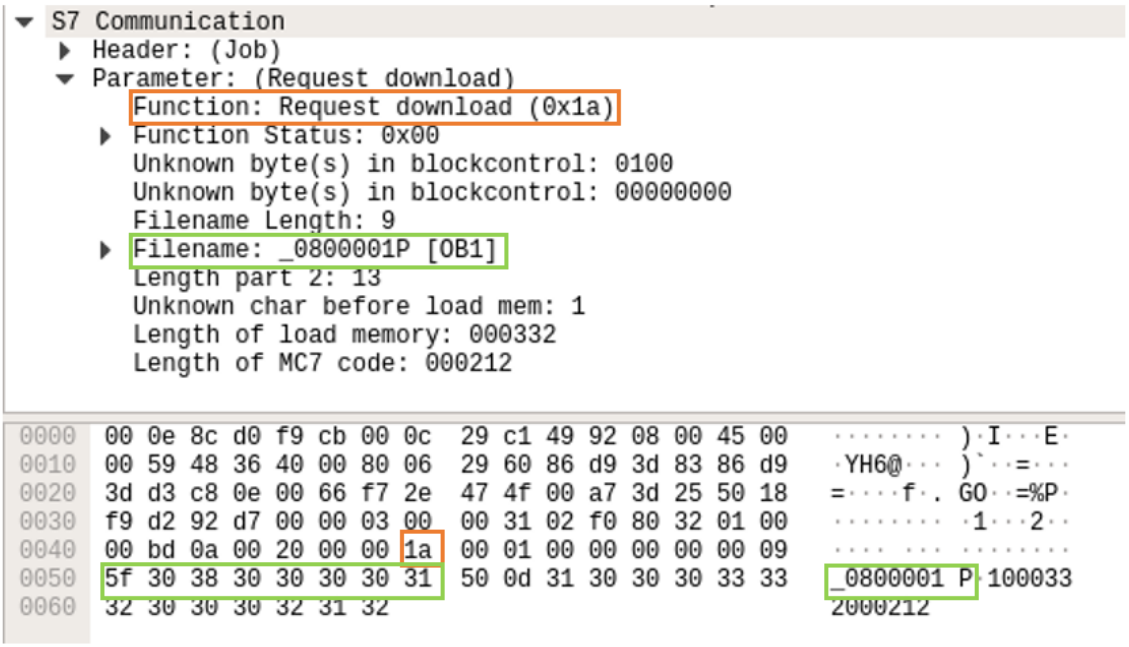

Figure 2.2: OB1 block download request in Wireshark

Something similar happens with PLC *Service Stop*, the first byte describes a PLC Stop message (0x29), while the last one contains the function name "P_PROGRAM" which sets the run state of the device. As can be seen in Figure 2.3

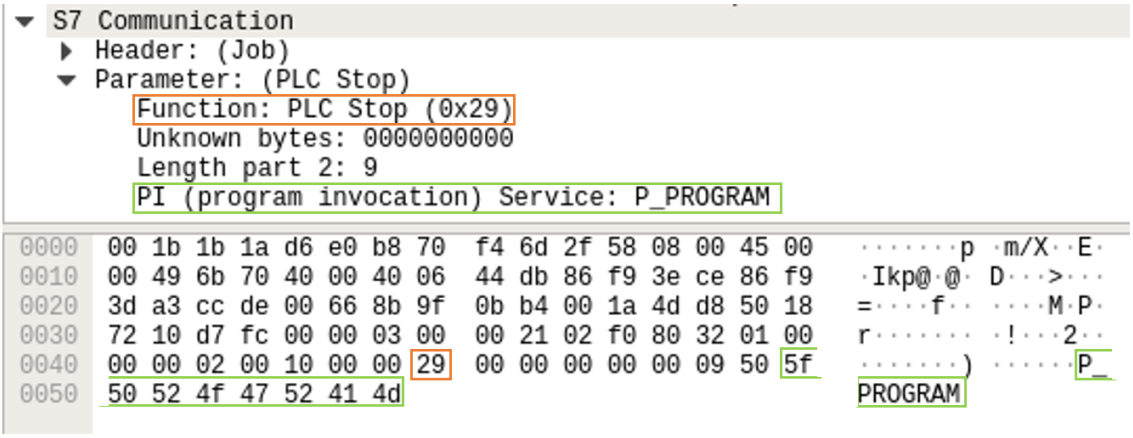

Figure 2.3: PLC Stop in Wireshark

S7Comm Security

- No integrity protection;
- No confidentiality protection;
- message injection and modification is possible;
- A more secure implementation of the S7Comm protocol has been implemented - S7Comm plus.

IEC 61850 - MMS

As described by Petr Matoušek in [41], IEC 61850 is an international standard for ICS communication, most commonly employed in environments pertaining to electric power systems. To exercise a client-server model on communications between the devices (PLCs, RTUs or IEDs) and the control station, the standard describes the Manufacturing Message Specification (MMS) protocol, for that effect.

MMS is a messaging system responsible for modeling real device communication and it's functions, and for transferring process data and information related to the devices in real time and also exchanging control information between applications and network devices.

As stated, IEC 61850 employs a client-server model through MMS protocol. The client is, typically, a monitoring system or control center that requests data or an action from the server. The server is a device that contains a VMD (Virual Manufacturing Device) where process variables, services and logic is stored (VMD objects), which is accessed by the client.

The MMS protocol is divided into two types of communications: Confirmed MMS services and Unconfirmed MMS services. *Confirmed-Request* PDU is the mechanism by which Confirmed MMS Services are requested, *Confirmed-Response* PDU and *Confirmed-Error* PDU depict the response sent to those requests. IEC 61850 defines 14 types of MMS PDUs, identified by their TLV (tag-length-value) as is portrayed in Table 2.7. Protocol data is encoded in a TLV triplets structure.

Confirmed MMS messages (*Confirmed-Request* and *Confirmed-Response*) are the workhorse behind MMS communication. Every *Cofirmed-Request* PDU contains a unique identifier that represents the type of the request: *status*; *getNameList*; *read*; *write*; among others, to wich we can designate as services. The types of services associated with Unconfirmed MMS messages fall on *Unsolicited Status*, *InformationReport*, *EventNotification* and *additionalService*. Concluding MMS messages are used to close the connection to the server and release the resources used to establish it. In [60] it is possible to se an abstract syntax of all MMS messages.

With regards to encapsulation, MMS is encapsulated over TCP on port 102 which can be beneficial for parsing purposes. Nevertheless, encapsulation includes several other ISO protocols like

TLV identifier	MMS PDU
0xa0	confirmed-RequestPDU
0xa1	confirmed-ResponsePDU
0xa2	confirmed-ErrorPDU
0xa3	unconfirmed-PDU
0xa4	rejectPDU
0x85	cancel-RequestPDU
0x86	cancel-ResponsePDU
0xa7	cancel-ErrorPDU
0xa8	initiate-RequestPDU
0xa9	initiate-ResponsePDU
0xaa	initiate-ErrorPDU
0x8b	conclude-RequestPDU
0x8c	conclude-ResponsePDU
0xad	conclude-ErrorPDU

Table 2.7: Types of MMS PDUs

TPKT and COTP among other ISO TSAP oriented protocols. Usually the first two encapsulations correspond to TPKT headers, given by this sequence of bytes `|0x03 0x00 0x?? 0x??|` (where "??" signifies that it could be any character), and COTP headers, given by this sequence of bytes `|0x02 0xf0 0x80|`. It is important to note that not all the protocols will be present in every MMS message, which adds a certain degree of uncertainty to the exact location of the MMS PDU payload.

IEC 61850 - MMS Security

- There is access control through *accessControlList* objects according to the ISO standard that dictates which client can delete or modify objects;
- No protection against replay attacks - An attacker may sniff the network to obtain an MMS packet and replay it at an unsuitable moment in industrial process, effectively disrupting production;
- According to [67] there are no other security mechanisms in MMS.

2.2 Intrusion Detection Systems

Intrusion Detection systems are responsible for monitoring a network and its devices in order to detect attacks and/or intrusions, through traffic monitoring and host behavior, that could jeopardize the overall functioning of the system. They are fundamental in providing intelligence and visibility on the network.

As previously mentioned, the scope of this dissertation covers the OT setting, rather than the usual IT setting, where IDS solutions are more commonly implemented, and thus, there are different requirements and challenges for when trying to deploy an IDS solution in industrial systems [29].

Importance of IDS

When configuring IDS tools, it is extremely important to establish rules that can distinguish malign traffic from benign traffic precisely and without being too comprehensive. The incorrect classification of traffic could lead to an overabundance of “false positives”, where traffic is incorrectly flagged as malicious. This poses a problem when later processing logged records as it becomes difficult to distinguish between a real attack and an incorrect alert. Assuming that the established rules do not promote ”false positives”, it is also of great importance do detect attacks as quickly and as efficiently possible.

For ICS operators, the inability to monitor and control critical functions and processes is the main cause for concern, since devices can become damaged beyond repair without operators noticing or being able to do anything about it [29]. A halt in production could lead to enormous amounts of revenue being lost per minute of downtime, and, in case of equipment damage, human lives could be put at risk by resulting faults [29]. To further solidify the importance of avoiding false positives and promoting detection efficiency as much as possible, the fact that it becomes difficult to distinguish malicious traffic from benign traffic while also being too slow at detecting intrusions, can result in a loss of view (inability to monitor critical control functions), which can, down the line, cause a loss of control (inability to affect the process), when a nefarious actor successfully compromises the system, two concerns brought upon in [29].

Taking this into account, it is of special importance to select the best type of IDS (discussed in Section 2.2.1), the best deployment strategies (discussed in Section 2.2.2) based on the network requirements and needs, and the best IDS implementation (elaborated in Chapter 4) to detect intrusions with utmost efficiency while also ensuring the least amount of false positives.

2.2.1 Intrusion detection types

Intrusion detection categories

Depending on where detection takes place, IDS can be classified in two different categories.

Network-based intrusion detection (NIDS), which are IDS running on a particular host, strategically positioned on the network, that is responsible for analyzing traffic between all network components and triggering alerts if any suspicious behavior has been seen [12].

Host-based intrusion detection (HIDS) which are IDS running on the hosts or devices in the network, monitoring inbound and outbound traffic, process execution, system memory and alerting on suspicious behavior [12].

The latter are difficult to implement in an ICS/SCADA environment as the components found in these systems are not as easily configurable as standard IT components, but the use tendency is to grow [29].

Intrusion detection methods

There are also different methods to detecting intrusions through HIDS and NIDS: Knowledge-based intrusion detection and Behavior-based intrusion detection.

Knowledge-based intrusion detection is employed by using knowledge gathered from previous known attacks and vulnerabilities and then spotting the re-occurrence of these exploits. This knowledge can be reflected through rules, which describe the attacks semantically, and signatures, which transform the semantic description of attacks into information that can be interpreted as a sequence of events that constitute an attack scenario [12]. These types of IDS methods are also referred to as signature-based IDS.

Behavior-based intrusion detection requires the establishing of normal system behavior and its components to then spot deviations from its expected behavior – this can help detect unknown attacks. The use of statistics among these kinds of implementations is relatively common [1], [2] and is done by sampling different system variables over time, establishing standard variable deviation, and then detect if some deviation threshold has been exceeded [12]. Machine learning and Neural Networks are also a common approach to implement these detection methods, trough the use of algorithmic techniques it is possible to train a model in order to learn the behavior of actors in the system, and, given new inputs (possible attacks), detect whether or not it constitutes an intrusion. These types of IDS methods are also referred to as anomaly-based IDS.

2.2.2 IDS deployment strategy

From an overall ICS environment perspective, including the enterprise and DMZ segments of an ICS network, it is important to identify all possible internet and extranet connections; identify remote access entry points, if any; and identify the different segmentation boundary points, to have a general understanding of the networks necessities and limitations. Subsequently, critical assets must be defined in order to understand where to focus security resources, like the IDS sensors, in this instance.

As previously mentioned in Section 2.1.3, the ICS networks are usually divided in segments,

and, as such, an ideal IDS placement should be balanced, covering IT (enterprise-network and DMZ segments), OT (industrial control network segment) and physical boundaries [29]. Studies from Longe Olumide et al [2] and Nicholas Pappas [52] touch upon this topic, from an IT perspective. Some extrapolation can be made to accommodate ICS notions.

Deployments

There exists multiple ways in which an IDS can be connected in order to monitor network traffic. Three of the most discussed will be examined below.

Inline: In this instance (Figure 2.4), an IDS is connected at the boundary between an internal network and an external network. Its operation is similar to a Firewall, given the fact that this method is commonly used to block network traffic - which we can refer to as an Intrusion Prevention System (IDS). This may prove beneficial with regards to intrusion prevention, but poses a central point of fault. If the IDS goes down, so will the connection between the dividing networks. Although this is troublesome in an IT setting, the severed connection between the industrial control network and the enterprise-network, for example, may not be as problematic from an ICS perspective, since the industrial process can still operate, albeit with a loss of view and remote control, which in itself is another issue altogether. Furthermore, issuing a preventive type of control, upon wrongly characterizing benign behavior has malicious, could affect normal industrial process. As described in [52].

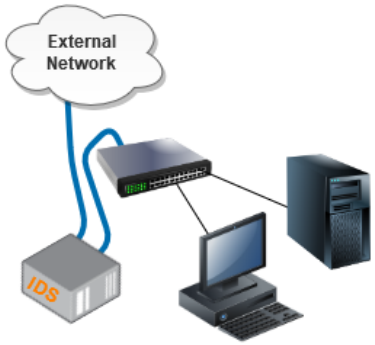

Figure 2.4: IDS sensor placed inline

Spanning Port: In this example (Figure 2.5 an IDS sensor will be connected to a switch, with a spanning port configured, so that traffic passing trough the switch is copied and sent to that port, which is where the IDS will also be connected to. This method is among the most common. It is generally easy and inexpensive to implement, since it only requires some pre-configurations to an already existing switch. From an ICS standpoint, it may not be that simple. From what has already been discussed, regarding security limitations and the static nature of ICS, it is possible to infer that, on an industrial control network, switches would not have a spanning-port pre-configured. Configuring the switch after setting up the industrial control network could

induce a halt in process. If this is not the case and the switches are, indeed, configured to copy incoming traffic to a different point, this could be the best option with regards to monitoring and detection, since it does not influence the overall process. As described in [52].

Figure 2.5: IDS sensor placed in spanning port

Network Tap: A network tap (Figure 2.6) replicates all data passed through the wire to the IDS physically connected to it, my means of cable rearrangement. This is not the best deployment approach, given the fact that the network taps must be acquired later on and it's installation will surely entail network interruption. As described in [52].

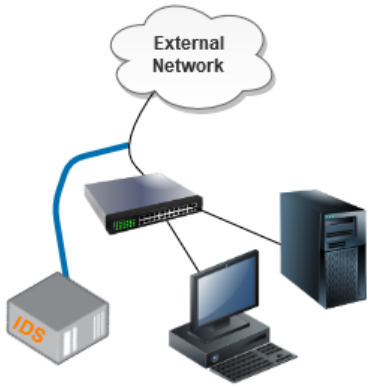

Figure 2.6: IDS sensor placed in network tap

2.2.3 IDS solutions

Some of the most common intrusion detection tools will be discussed in this section. Providing some contextualization to what open-source solutions exist.

Snort

Snort [7], [1], designed by Martin Roesch in 1998, is a lightweight Network IDS (NIDS) that relies on pattern matching rules to detect dubious traffic and possible attacks on a network. When con-

figured, the software generates an alert when a possible match is found between the traffic being monitored and the created rules. With newer versions of the software, some decoding modules for ICS protocols, like Modbus and DNP3, have been added [34], allowing for the creation of rules better suited for ICS environments.

Suricata

Suricata [65], owned by the Open Information Security Foundation (OISF), is a robust network intrusion detection and prevention software, that also monitors traffic on a network. Like snort, it also relies on the creation of rules and signatures to employ pattern matching while also supporting its own scripting language to help detect complex threats. Its input and output formats promote a better integration with databases and other monitoring tools, like SIEMs (discussed in Section 2.3). It is very focused on security, usability and efficiency.

Bro/Zeek

Conversely, Zeek (formerly known has Bro IDS) [69], developed by Vern Paxson and then maintained by Robin Sommer, is an extremely flexible network analysis framework, not only focused on intrusion detection and monitoring but also in providing a platform for network traffic analysis. Possesses an adaptable scripting language that enables the implementation of site-specific monitoring policies. Does not rely solely on the creation and manipulation of signatures and rules, but can be used to that end, leveraging its signature framework. Part of the power behind Zeek is the rich amount of logs it produces.

2.3 Security Information and Event Management

Security Information and Event Management systems (SIEM) were designed to help correlate events, logs and other information generated from several monitoring tools and normalize and aggregate this information, in order to make a correct assessment on potential threats [4].

A SIEM system is responsible for centralizing all security notifications and alerts from various security tools implemented on a network. All these tools generate an enormous amount of logging data, which becomes difficult to process locally. The SIEM gathers all this information and aggregates it under one solution.

More importantly, is the ability to ensure automated cross correlation and analysis of all the raw event logs from the entire network. A SIEM then looks for security issues that would otherwise go unnoticed if only looked at from the isolated perspective of each of these tools

A well configured SIEM solution understands what applications are running on which network devices in order to contextualize and interpret what alerts could be false positives. Adds critical context to all the events and notifications received, which allows the SIEM to notice changes to critical devices.

2.3.1 Log Analysis

Logs

A log file is a file where system, application and device events, as well as messages between these components, are recorded. It is a mechanism used for posterior analysis, providing an audit trail that is used to understand the behavior of every network and system component. It is particularly useful from a security perspective, as it could be leveraged to obtain important information regarding potential threats.

A SIEM can receive logs from diverse sources [4], like firewalls, switches, routers, servers, historians, devices, IDS alerts and other software metrics and performances. This, along with other important data, is imperative for threat assessment.

Telemetry data

Telemetry refers to the collection of measurements on remote or inaccessible points and the subsequent transmission of this data to a system in a different location, responsible for monitoring and analysing this data [63], [66]. In this scope, we could argue that field devices and other components would be relaying this data (status, temperature, pressure, etc..) to a SIEM.

There are sensors at the remote points that measure electrical and physical data which are then combined with timing data, forming a data stream to be transmitted to the monitoring system. A SIEM, in this case. When this information is received by the monitoring system, it is desegregated, obtaining the original data separately to be processed.

This is helpful, from a threat detection standpoint, as this data becomes very important in understanding the status of the devices communicating through the network, and understand if everything is running according to an established baseline.

Heuristics

Heuristics are strategies that are established to help form judgements for solving adaptive problems [38]. We could, for instance, establish a baseline for all communication and expected component

behavior:

- Compile list of devices connected to ICS network
- Describe baseline communication

    - Rate of Success

    - Amount of data for each session

    - Type of Data

    - Know what addresses can access the network

    - Understand balance of network traffic (Average load per component; how often critical data is stored; internal vs external traffic; etc)

    - Peak performance vs normal conditions

This information must be documented to be later accessed by a monitoring system in order to detect possible deviations on what is currently happening on the network. That could be the SIEM's responsibility.

Information Correlation

After establishing all *heuristics* and obtaining all the *logging information* and *telemetry data*, this information must undergo some analysis. The SIEM aggregates all this data and is able to link the diverse data types in order to employ some kind of correlation, through a correlation engine. This allows a SIEM to find potential threats in the network and issue alerts and reports. For example, given some device state change and the appearance of an unknown address on the network, a SIEM would assert if this could be cause for concern. Figure 2.7 Illustrates a SIEM architecture.

2.3.2 SIEM solutions

Elastic Stack

Elastic [14],[15] provide a stack of tools that, allows for data collection and from any source while later allowing for the filtering of this data. Resorting to the different tools, a user can establish a command center by creating dashboards that provide data visualization. If correctly deployed and configured, a user can detect issues that may arise on the network being monitored.

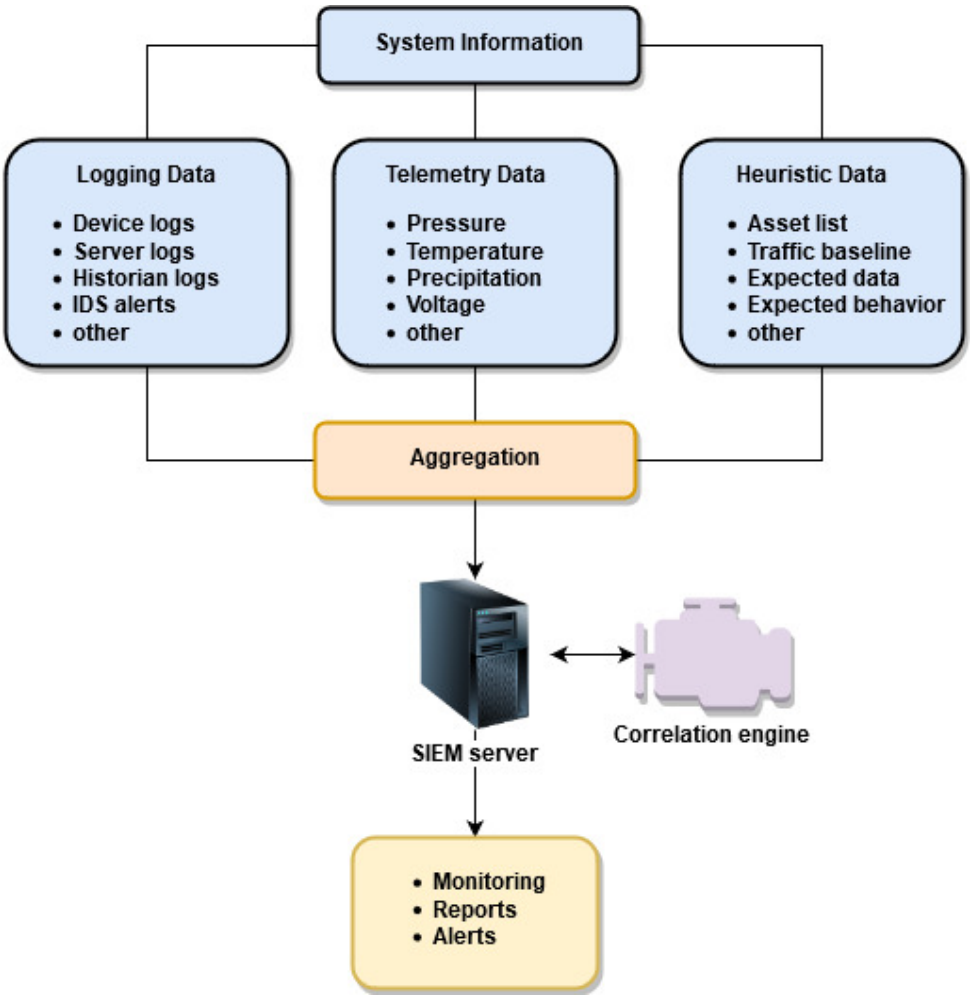

Figure 2.7: SIEM architecture

Graylog

Graylog [22] is a platform for centralized management topically used as a SIEM. It can gather large quantities of different data inputs to be processed. Its correlation engine allows for the creation of complex alerts based on the relationship between different logs, alerts, events and conditions collected from the environment. It also provides a dashboard function to allow for a better visualization of displayed data.

Splunk

Splunk [13], constitutes a platform that is able to poll data from devices from ICS networks (PLCs, HMIS, Historians, DHCP/DNS servers, etc..) in order to detect deviations or issues regarding device behavior and communications. Provides real-time monitoring of known, unknown and insider threats [31].

2.4 Related Work

Andrew Nicholson et al [47] developed an early warning system to be deployed on an ICS/SCADA environment by establishing runtime-monitoring techniques based on a Interval Temporal Logic framework.

Dina Hadžiosmanovic et al [24] present a detection solution that can track updates to process variables (by extracting variable updates from network devices) and construct variable specific prediction models to help establish a baseline for future activity analysis.

William jardine et al [32] propose an implementation of IDS (SENAMI) that monitors networks passively with selective non-invasive active monitoring on specific ICS attack vectors. The work highlights the deficiencies of passive Network IDS (NIDS) alone for detecting targeted ICS cyber attacks. Their main concern is to avoid computational and network overhead by polling a limited number of contextualised variables from target components. This ”active non-invasive” approach compares particular PLC variables to detect a specific ICS attack use-case, tempering of monitoring information to the control process. This work is tailored for Siemens specific components, which may prove beneficial from this dissertation standpoint. Related to the topic of gathering device information, Mislav Findrik et al [20] propose an implementation of PLC logic data logging called PLCBlockMon, demonstration its applicability for intrusion detection in specific scenarios while also providing guidelines for its usage.

Most related work found for safeguarding ICS/SCADA systems revolve around Intrusion Detection. This is an obvious choice for securing and monitoring critical infrastructures, although, other security controls should also be employed to help safeguard these systems.

Some technical reports [31],[25] were found that outline some ICS deployable security tools. Rubio et al [57] present an article that studies the spectrum of attack vectors used by advanced persistent threats (APTs) against industrial ecosystems in order to understand what defense mechanisms can be used as a first line of defense, but they culminate by providing an analysis of the evolution and applicability of IDS.

2.5 Security notions

This section depicts some other notions with regards to Security concepts that may be fundamental to understand the topics talked about for the remainder of this dissertation. Explain some cyber attack and general concepts

[where else to put this extra info?]

2.5.1 Phishing attacks

Phishing is a type of attack that is related to social engineering (the act of manipulating people to give up confidential information). It is through phishing, that attackers can either obtain sensitive information, install malware on a victims computer or crashing the computer altogether. The victim is sent a malicious attachment on an email or a message and is tricked into opening said attachments or login-in to fake websites made to look like the real thing, effectively installing a malware on their computer or even give out their sensitive information unknowingly.

This topic is important in the scope of this dissertation, as it represents the first time the attacker may come in direct contact with the corporation he is trying to compromise, ultimately achieving his final goal of disrupting the industrial control network [54].

2.5.2 Denial of Service

A Denial of Service (DoS) attack is an attack aimed at forcing an unavailable condition onto computers and devices effectively stopping or shutting them down [9].

DoS can either over-saturate the capacity of the targeted host by flooding it with an overwhelming amount of network packets, effectively turning the host unresponsive from so many unprocessed packets. Furthermore, an attack on which the hosts resources are completely consumed by overflowing the hosts memory can also achieve a DoS condition.

Achieving a denial of service condition is, arguably, the main objective of threat actors with regards to critical infrastructures. The goal is to disrupt industrial process, forcing control devices to shutdown or become inoperable. Thus, the importance of detecting and preventing denial of service attacks is paramount.

2.5.3 ARP poisoning

The address resolution protocol (ARP) is utilized to map IP network addresses to the machine's hardware addresses (MAC address).

ARP poisoning is a mechanisms by which an attacker sends spoofed ARP messages to associate the attacker's MAC address to an IP of a benign host of the network, which would result in every traffic meant to that host to go to the attacker's machine instead. This attack is used to establish a Man-in-the-Middle position between two hosts. It's importance in the ICS scope, will be explained further in Section 4.8.5.

2.5.4 Metasploit

The Metasploit project, created in 2003 by HD Moore, is a platform that helps understand how secure a computer is on a specific network. With Metasploit, attacks to security flaws on diverse software and hardware components of computers. This tool is constantly updated with more mechanisms to exploit security flaws discovered by professionals.

This tool was developed to help security researches understand how insecure their systems were, so they could improve security deficiency, but it is particularly useful for attackers as they obtain the same tool-set to ruin systems by expliting their vulnerabilities.

2.5.5 Ettercap

Ettercap is an open source tool suite developed by the Ettercap Dev. Team, ALoR, NaGA that can be used for security auditing by esbalishing a man in the middle position between hosts. This tool is often used by attackers.

2.5.6 MIME

A media type (also known as a Multipurpose Internet Mail Extensions or MIME type) is a standard that indicates the nature and format of a document, file, or assortment of bytes. It is defined and standardized in IETF's RFC 6838.

Chapter 3

Analysis of industrial attacks and security mechanisms

This chapter presents the thought process, conducted tests and further research that helped formulate a decision on potential security mechanisms to protect industrial environments from threat actors. Firstly, a fundamental insight regarding ICS attack route taken by an attacker, followed by an analysis of security mechanisms and how they could, or not, be useful from an industrial defense perspective, culminating on a potential solution for protecting ICS environments, which will be a focal point for the remainder of this thesis.

3.1 Anatomy of an ICS attack

This section depicts an example of a possible path taken by threat actors with the goal of compromising an industrial system. Starting from the reconnaissance phase, prior to exploiting the corporate network, all the way to finalizing the attack at the industrial process environment. The example depicted in this section was taken from a security awareness video [58].

3.1.1 Reconnaissance

In the first reconnaissance phase, attackers study the potential entry-ways into a company's network by leveraging public information or social engineering (referred in Section 2.5.1). They could start by finding lists of preferred vendors; Company asset locations publicly registered and an in depth analysis of the organization's employees by taking advantage of social media and the company's website.

3.1.2 Gaining access

To gain access to the organization's enterprise network, attackers can perform a phishing attack (detailed in Section 2.5.1) with a corrupted attachment that exploits a vulnerability on a document rendering software installed on the employee's workstation, infecting the workstation in order to install a command and control server (detailed in Section 4.8.2); Having compromised the workstation, attackers use it as a staging platform inside the organization's network, uploading customized tools from their command and control channel and performing further reconnaissance on the organization's network.

3.1.3 Reconnaissance (after compromise)

Threat actors are, by this point, able to identify internal application servers accessed by employees; read corporate emails; compromise other systems; backup local files from the compromised systems and exfiltrate that information (normally trough the established command and control channel); install key-loggers to capture login key strokes and obtain credentials. And, most importantly, discover a remote access server that will allow them to pivot to the industrial control network - their main goal.

3.1.4 Persistence

Spread access to other computers; erase log files of their actions to obfuscate their presence; compromise key operational systems. Use the same commands that the administrators use to obfuscate presence, continue to exfiltrate data. By previously installing key loggers it might be possible to gain administrator access to the Active Directory used for authorization and authentication to the read-only corporate historian server in Corporate Network, granting additional permissions to compromised user accounts and then modify groups to provide access to sensitive systems. By now, the attackers possess multiple points of entry and exfiltration; established a foothold on numerous workstations and have freedom of movement.

3.1.5 Execution

By monitoring traffic to and from the read-only historian server, attackers identified that there was a master historian server on a Supervisory Control Network at the ICS layer, connected to the replicated historian (bi directional trusted connection) in the Corporate Network. Injected data on the replication process of the read-only historian, infecting the master data historian with malware. This malware would be responsible for bombarding the network with excessive garbage data and other network exploit attacks. Resulting in a loss of view in the HMI and a fault condition on a

remote PLC by one of the many exploits carried out on the previous phase.

With this attack pathway example, one can understand how an attacker proceeds in order to establish a foothold on the industrial network in order to disrupt industrial process. With this in mind it is imperative to try and respond to any intrusions happening across each of these phases.

3.2 Survey of security mechanisms

In this section some fundamental aspects regarding work efforts and research conducted will be presented with the goal of deciding on a potential security mechanism to deploy on industrial control systems.

3.2.1 IDS placement

While studying the possibilities for the deployment of security controls to protect ICS settings, some decisions had to be taken regarding IDS placement. As mentioned previously in Section 2.2.2, placing either an IDS or IPS *inline* can run the risk of bringing down the connection between two dividing networks [52]. Also, considering the priority of availability and avoiding, as best as possible, disruption of device communication, having a preventive solution (IPS) issue control over the network, after falsely identifying benign traffic as malicious, goes against ICS principles [52] [62] [12].

With this in mind, and also discarding the idea of implementing a *Network tap*, given its disruptive nature [52], opting for a *spanning port* approach should be the safest and less cumbersome option for an integrated intrusion detection and monitoring solution.

3.2.2 Pfsense

The PfSense project is an open source distribution, based on the FreeBSD operating system that serves as a network firewall. Third party software packages can be added to this distribution, effectively turning it into an intrusion detection and prevention system.

PfSense software comes with a web interface that allows for the configuration of its components - acquiring new packages and configuring firewall and IDS rules.

On a relatively primordial state of research, some tests were conducted to understand the possible applicability of a solution like Pfsense. To that end, a network example was set up.

- The pfsense firewall was configured as a bridge between two other machines (in *inline*

mode), containing both Snort-IDS and Suricata-IDS along with some firewall rules for traffic redirection.

- A monitoring machine, responsible for configuring Pfsense through its webserver. The host was connected to the Pfsense firewall through a local area network (LAN) interface.
- An attacker machine, with the metasploit attacking tool (Section 2.5.4) installed, was used to conduct available ICS specific attacks to the victim machine. The host was connected to the Pfsense firewall through a wide area network (WAN) interface.
- For a visual representation of this simple architecture, Figure 3.1, portrays what was discussed

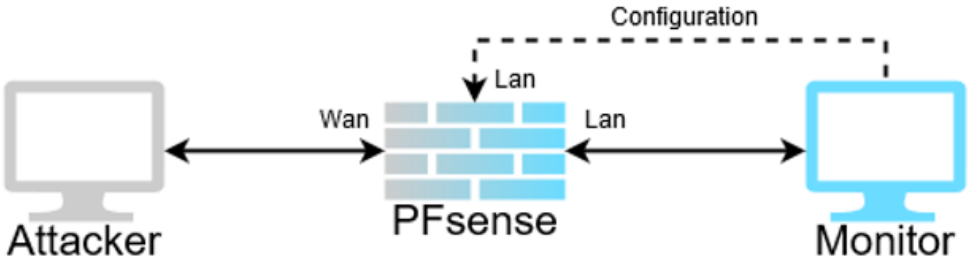

Figure 3.1: pfsense architecture deployment

On Pfsense, Snort and Suricata intrusion detection systems were configured. On these IDS, some ICS rules were deployed in order to test how Pfsense would react to attacks specific to ICS, most of which were detected. Thus the same attacks and corresponding detection rules were carried over for testing in the final implementation of this thesis solution (Chapter 4).

This idea was not embraced fully because its implementation requires that the Pfsense host be deployed in *inline* mode. As explained earlier in Section 3.2.1, this is not the foremost way of deploying an intrusion detection/prevention and monitoring system, to help protect Industrial environments.

3.2.3 SiLK

System for Internet-Level Knowledge, SiLK - developed by the *CERT Network Situational Awareness Team* (CERT NetSA) - is a suite of traffic analysis tools that allow security analysis of network flow data. It is a very powerful tool that enables security analysts to gather historical data and parse this data through the use of available commands. Although SiLK is a great tool for flow data analysis, it may not be a good choice with regards to implementing an integrated intrusion detection and monitoring solution for ICS networks, since the detection and prevention of threats must be done as quickly as possible. SiLK only provides a way to analyse historical data.

A potential way of improving the value in using this tool would be to automate the detection process by creating a program (for instance, using python) that would call the SiLK suite to analyse flow records. By knowing how attacks take place, and how these attacks influence the network flow, we can implement an automated program that enforces this analysis automatically without the need for manual intervention. For example, If an attack works by having a machine sending several requests to open TCP connections through different attack send frequencies and duration time, we could create a program that would:

- Call SiLK suite to divide the flow records by source IP
- Assess which of those IPs sent requests through TCP
- And, for each IP, how often was the frequency and duration through which they sent those requests

Although it is an interesting approach, a straightforward flow analysis program may not be enough to deal with more specialized intrusions that do not influence network flow. Furthermore, as previously stated, only being able to enforce detection mechanisms after network communication took place, is not the best way to undertake security detection issues. Another approach must be considered.

3.2.4 Yara

Yara is an open source tool designed to help in the detection of known malware by exploring code similarities between malware samples based on textual and binary patterns, which are described when creating Yara rules. Yara rule files include signatures that describe the malware's byte-sequences and/or strings to which it would trigger an alarm. It runs on hosts (multi-platform) where the malware could be found in, analyzing each file in the path configured to be monitored.

In some respect this approach is very beneficial from an ICS perspective, as it can help identify the presence of known malware from important industrial attacks carried out this past decade (ICS attacks go as far as 1993 with the Marconi Wireless Hack) (Table 3.1 depicts some examples of Yara detectable malware). However, according to Slowik [62], attackers are changing their attack approach by deviating from the use of custom malware and tools while relying more on using the own system's tools against it (to what he calls "Living of the Land").

ICS malware	Year of occurrence	Yara Rules
Stuxnet	2010	https://github.com/Yara-Rules/rules/blob/master/malware/APT_Stuxnet.yar
Dragonfly/Havex	2013	https://github.com/mikesxrs/Open-Source-YARA-rules/blob/master/US%20CERT/Dragonfly.yar
Blackenergy2	2015	https://us-cert.cisa.gov/sites/default/files/file_attach/\acs{ICS}-ALERT-14-281-01E.yara
CrashOverride	2016	https://us-cert.cisa.gov/sites/default/files/file_attach/\acs{ICS}-ALERT-17-206-01.yara
Trisis	2017	https://github.com/MDudek-\acs{ICS}/TRISIS-TRITON-HATMAN/blob/master/yara_rules/ics-cert-v2.yara

Table 3.1: ICS malware attacks with Yara support

Malware hashes and signatures depend on the overall environment conditions to which malicious software focuses on. This software is seldom replicated and reused on other attack campaigns [62]. Therefore, this approach will likely fail to detect new attacks that are not carried out in the same conditions as the, available, malware hashes and signatures were.

However, this approach adds surplus value to any security implementation for industrial control systems. Even though it was not implemented nor explored further for this dissertation.

3.2.5 Iptables

Iptables is a powerful tool that acts as a firewall for linux-based systems. Originally developed by Paul "Rusty" Russell, it works with the linux kernel, to manage network packets going to and from the system in which it is installed. Iptables block, accept or take other actions based on user-defined conditions. These conditions are called rules, which are configured to enforce control policies and manipulate network traffic. Network packets will be checked against the rules, by order in which they were deployed, to then apply some action.

The Iptables firewall is extremely useful for blocking unwanted or nefarious traffic, and should be considered by most implementations where linux-based systems are predominant. And yet, it may not be the best choice for defending OT environments.

To guarantee network control through a linux-based firewall, one must be part of the ICS network architecture, which isn't too common on industrial environments. Usually, ICS utilize industrial firewalls with proprietary software, therefore it is unlikely that Iptables are the foremost tool utilized for network traffic control. On circumstances where there are linux-based systems enforcing control policies on traffic that pivots from the IT environment to the OT environment and when linux-based systems are set at the boundary between two ICS segments it could be enticing to implement such a firewall, yet, this seldom occurs.

At some point in the development of this thesis main security solution (introduced at the end of this Chapter and later expanded in Chapter 4), some effort in implementing a mechanism for automatic generation of Iptables rules went into this topic, although it was scrapped for not seeing too much applicability.

By leveraging Zeek-IDS *Exec Utility* [68] to call outside code from a running Zeek policy script, it was possible to, on a triggered alert, create a new Iptables rule to block traffic coming from the responsible IP and write it to an Iptables configuration file. This file would then be deployed on the linux kernel, not allowing further communication from the host responsible for triggering the alert. Since the linux system on which detection is running, has no direct contact with the network process, no definite action could take place.

3.3 Decision

All the analysed tools have an important role in protecting these systems and should be considered when deploying security measures on a real environment. By delving into, and discussing the different mechanisms and approaches for intrusion detection, network monitoring and system protection along with their prospects, a potential solution was considered - ICS-Security-Appliance. For this purpose, the original concept was to combine both **Snort-IDS** and **Zeek-IDS** tools in one integrated solution, a mechanism to baseline traffic behavior, **Bropy**, and a SIEM/Dashboard to visualize the potential intrusions and enforce some event correlation for better alert granularity, **ELK stack**. The idea behind combining both Snort-IDS and Zeek-IDS stems from a phrase in a thesis from J. Nivethan and M. Papa: "Our prototype takes advantage of the strengths of both Snort (network-level monitoring) and Bro (process-level monitoring)" [48].

At the time, the initial proposal was a mechanism that leveraged Snort-IDS and its signature based approach to detect known attacks based on rules, while also being able to use a detection method closer to industrial process and traffic behavior, which would be carried out by Zeek-IDS, as it possess a powerful scripting language and event engine to deal with traffic. Having both solutions monitoring traffic from a *spanning port* on a switch or firewall, after an alert from a potential attacker, the idea was to have Zeek-IDS scripting frameworks reconfigure the switch/firewall in order to block further communication from the host that triggered the alert, effectively protecting the network. Furthermore, after alerting on malicious behavior, the plan was to have the appliance produce a mitigation procedure to help security professionals deal with the problem in future occurrences. Every alert and relevant log would be shipped to ELK stack, acting as a SIEM, to conduct further analysis and promote visibility through visualisation dashboards configured in Kibana (one of the services of the ELK stack).

Figure 3.2 details the initial intrusion detection architecture for the ICS security appliance and its functionality.

It is important to note that, as will be explained further in Chapter 4, some accommodations and alterations had to be made to the initial proposal for the solution. Thus the implementation will differ from what was initially expected. Snort rules are no longer sent to the SIEM, there will be no switch/firewall reconfiguration nor an outputted Security Procedure after an alert is triggered.

A more detailed explanation of what has actually been implemented within the ICS-Security-Appliance is explained in chapter 4.

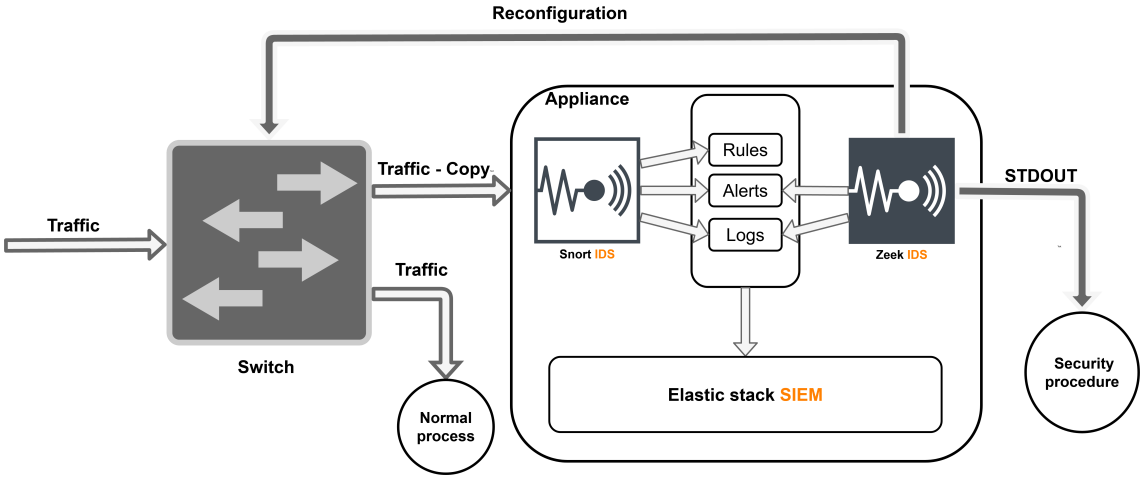

Figure 3.2: ICS-Security-Appliance intrusion detection and monitoring architecture

Chapter 4

ICS-Security-Appliance

In this chapter the fundamental aspects of the proposed solution, and it's implementation will be introduced. Starting with an overview of what has been implemented followed by a description of the base state of all the tools and mechanisms utilized in the development of the ICS-Security-Appliance (every default mechanisms not implemented by the author of this dissertation). After which a brief discussion on the decision behind the solution placement within the network, as well as which intrusion detection systems are going to be used. A section detailing the ICS-Security-Appliance's architecture and functionalities, a section detailing the solution's requirements and a section describing the attack model (what does an attacker have access to?). The following sections will be more focused on implementation, exhibit every characteristic behind baselining, detection use cases and the SIEM implementation.

4.1 Implementation overview

This section entails an overview of the work done with regards to the proposed solution.

4.1.1 Zeek for logging

Through Zeek's powerful event based scripting language it is possible to implement logging policies on traffic seen, allowing it to log virtually any information extractable from traffic. To that end, implemented policies include mechanisms that log communications from hosts that issued S7comm and IEC 61850 MMS instructions, log Siemens devices communicating on the network and log Modbus process variables. From Zeek's default policies, its ability to log every connection, DNP3 and Modbus communications was also particularly helpful.

4.1.2 Bropy for baselining

For traffic baselining purposes, based on the Bropy tool, several scripts were developed on top of it, in order to generate files containing information on traffic behavior. These files will be later accessed by Zeek, to compare new traffic against what has been registered to be benign behavior. Baselining mechanisms leverage logs previously created by Zeek. These mechanisms include ways to baseline which Siemens devices exist on the network; which hosts issue S7Comm and IEC 61850 MMS control instructions and what are those instructions; which hosts issue DNP3 and Modbus commands and what are those commands; A static route-table from observed traffic. From Bropy's default configuration, a mechanism for baselining Who talks with whom was also especially useful.

4.1.3 Zeek for detection

Zeek policies were implemented to deal with potential network intrusions, enforcing network-level analysis and process-level analysis. Detection of potential intrusions and attacks include mechanisms that verify the occurrence of ICMP and DNS shells; Man-in-the-Middle activity; discovery techniques for some Siemens devices; the usage of default credentials on the network, and credential spraying; Executable files being downloaded to the network; DNP3, Modbus, S7Comm and IEC-61850-MMS instructions out of the ordinary or issued from hosts not usually responsible for issuing them; Modbus and DNP3 dangerous functions; S7Comm service stop; program logic download and upload. With the help of default Zeek scripts for network scans, it was also useful for detecting port scans. Furthermore, Bropy traffic baseline deviation policy was also helpful in the detection of unusual endpoint communication.

4.1.4 Snort for detection

Using snort for network-level monitoring based on a rule-based approach, some rules were deployed on the Snort tool implementation. These rules help detect some dangerous protocol functions (specifically Modbus - Force Listen Only); Segment Smack exploit; a Siemens SiProtect DoS Attack through enabling update firmware mode indefinitely; Some metasploit attacks and other attacks for which ICS rules where found.

4.1.5 ELK Stack as a SIEM

Through the ELK Stack, a stack of different services, it was possible to implement a SIEM solution. For this SIEM, most of the important logs and alerts generated from Zeek and Snort were configured to be pushed into the ELK stack SIEM, for posterior analysis. Some visualizations

were also created, in order to promote visibility into the protected environment and allow security analysts to posses a context of the network being monitored. Visualizations include which alerts were triggered after an attack or intrusion and which hosts are responsible for triggering those alerts; A pie chart representing the ”top-talkers” of the network - who sends the most amount of bytes; A line graph detailing an average duration of communications; A search table portraying every alert with richer information.

4.2 Base State

In this subsection I introduce the base state of the solutions in their default conditions and which components are being used by my approach. Snort-IDS with deployed rules found online, Zeek-IDS with its default logging functionalities and the Bropy tool with its baseline python scripts and integration with Zeek-IDS.

Snort-IDS is installed with its default rules deployed, along with ICS rules found online.

- Quickdraw `https://github.com/digitalbond/Quickdraw-Snort`
- Talos `https://github.com/ITI/ICS-Security-Tools/blob/master/configurations/rules/talos-snort.rules`
- Codecat007 `https://github.com/codecat007/snort-rules/blob/master/snortrules-snapshot-29150/rules/protocol-scada.rules`

Bropy tool, developed by Matt Domko [28], is installed in the machine. Some of its functionalities are used to help baseline the traffic and detect deviations from it. From the components present in the tool, the python script **bropy_conparse.py** is responsible for reading Zeek connection logs and creating a baseline file that contains information on who communicates with whom. Also, **baselinereport.bro** is a Zeek script responsible for reading traffic and, on every new connection, compares it with what has been baselined in the previous baseline file. I took inspiration and help from the way this tool is implemented, to extend Bropy, in order to generate different baseline files from different Zeek logs, and also how to implement Zeek scripts.

4.3 Appliance placement

As previously discussed in Section 3.2.1, the *spanning port* approach is the foremost way of placing a security appliance of this nature. Its easier to configure and maintain while, also not interfering with industrial process, provided a spanning port can be configured on network devices. For this reason, the security appliance has been configured using this approach.

4.4 IDS decision

For the ICS-Security-Appliance, I came to the decision of having both Snort-IDS, as well as Zeek-IDS acting as defense mechanisms to alert on potential nefarious behavior.

- Snort - With Snort's signature engine and constant community support and development, it is relatively easy to deploy new detection rules as well as finding relevant rules being ever published by the community and other researchers - A significant amount of ICS rules are already available. Thus, ensuring an "up to date" defense for known attack vectors. Used for *network-level analysis*.
- Zeek - With Zeek's powerful scripting engine and rich logging framework it is possible to create and customize policies to deal with known ICS attack vectors, but, more importantly, malicious behavior that could take place in a specific industrial environment - a localized approach. In other words, Zeek's policies can be extended to better protect the environment where the appliance would potentially be deployed. Used for *network-level analysis* and *process-level analysis*.
- In the long run, new rules can be added to Snort, as they become available and new policies can be implemented on Zeek to ensure localized and specialized protection, depending on the industrial environment requirements where detection will take place.

4.5 System Model and Architecture

4.5.1 System Model

The ICS appliance encompasses both Zeek-IDS and Snort-IDS for *network-level analysis* and Zeek-IDS for *process-level analysis*. Both tools ensure detection by receiving a copy of the network traffic passing trough a switch/firewall and inspecting the network packets. Snort-IDS compares the network packets against rules/signatures deployed in its signature engine. Zeek-IDS, also receiving a copy of the network traffic, would check for specific strings that could pose a potential attack via pattern matching, while also being able to identify possible behavior deviations from a previously established baseline. Bropy, a tool with "Basic Anomaly IDS capabilities with Python and Bro" [28], is used and extended in order to establish a traffic baseline. ELK Stack receives all relevant log and alert data from the intrusion detection sensors for posterior analysis and event correlation

Connecting the ICS-Security-Appliance to a spanning port on a switch, the system model undergoes the following process:

1. Traffic is reproduced on the **Snort-IDS**, which produces alerts based on deployed signature rules;
2. Traffic is reproduced on **Zeek-IDS**, which produces traffic logs as well as alerts based on implemented logging and detection policies;
3. From previously generated **Zeek-IDS** traffic logs, **Bropy** generates baseline files that represent the network's benign behavior;
4. **Bropy's** generated baseline files are fed into **Zeek-IDS** in order to help produce behavior alerts based on implemented detection policies.
5. **Snort-IDS** and **Zeek-IDS** triggered alerts, as well as **Zeek's** generated traffic logs, are shipped to the **ELK-Stack**, acting as a SIEM, for posterior analysis and network visualization

Figure 4.1 depicts a diagram for the ICS-Security-Appliance system model. It helps to visualize the underlying functionalities of the solution, previously enumerated.

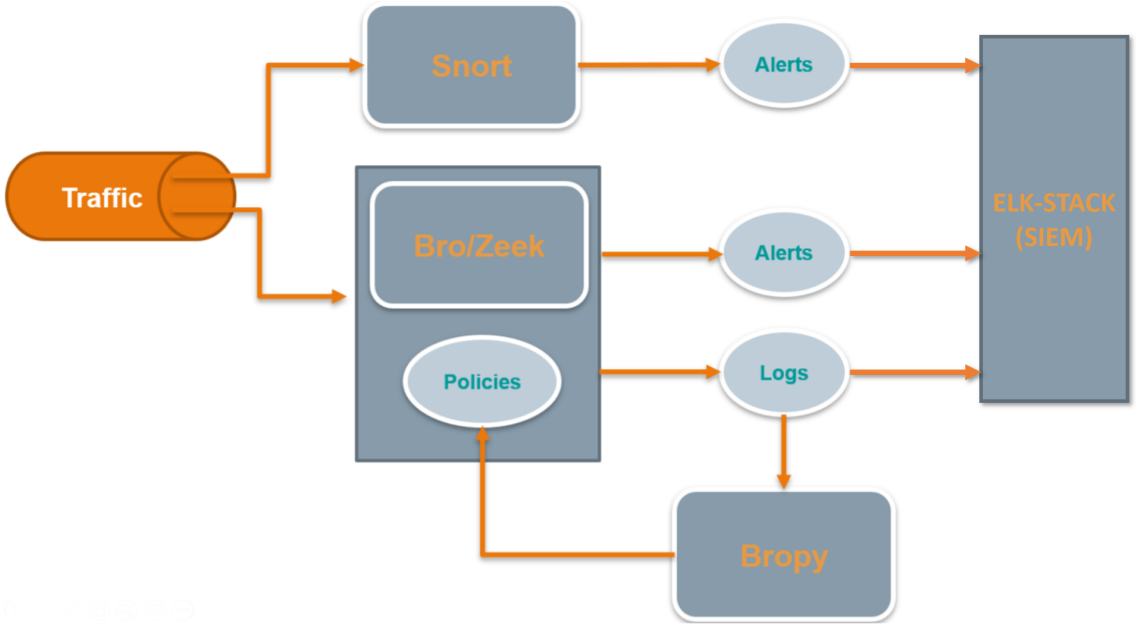

Figure 4.1: ICS-Security-Appliance system model

4.5.2 System Architecture

The environment constitutes a machine using a Linux(Ubuntu 18.04.5 LTS 64-bit) operating system with a network interface for management purposes and another network interface for sniffing and capturing traffic (where both Snort-IDS and Zeek-IDS sensors would be located in order to capture the traffic)

Figure 4.2 Entails the ICS-Security-Appliance placement in the industrial network architecture. It is imperative to have full visibility of the entire architecture, thus the need to have it placed in the Enterprise network, as well as the Industrial Control Network.

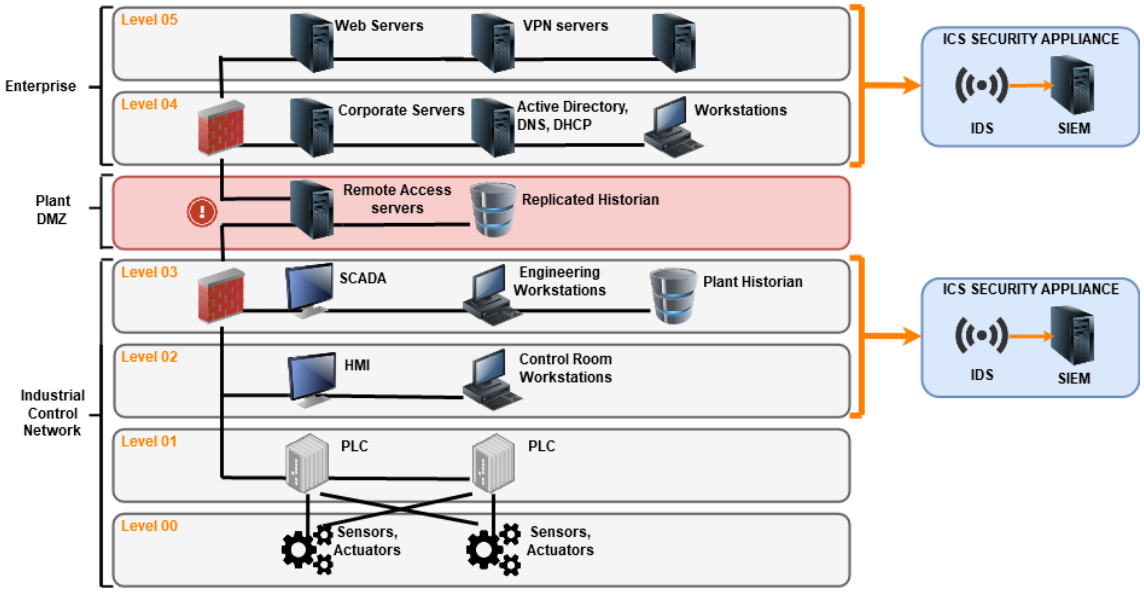

Figure 4.2: ICS-Security-Appliance placement within industrial architecture

At either of those layers, the ICS-Security-Appliance should be connected to a switch that grants the most visibility within those segments. Figure 4.3 portrays the revisited intrusion detection and monitoring architecture of the ICS-Security-Appliance as some of the functionalities referred in section 3.3 no longer hold true. There is no Switch reconfiguration nor security mitigation procedure being produced by Zeek and Snort rules are not shipped to the SIEM.

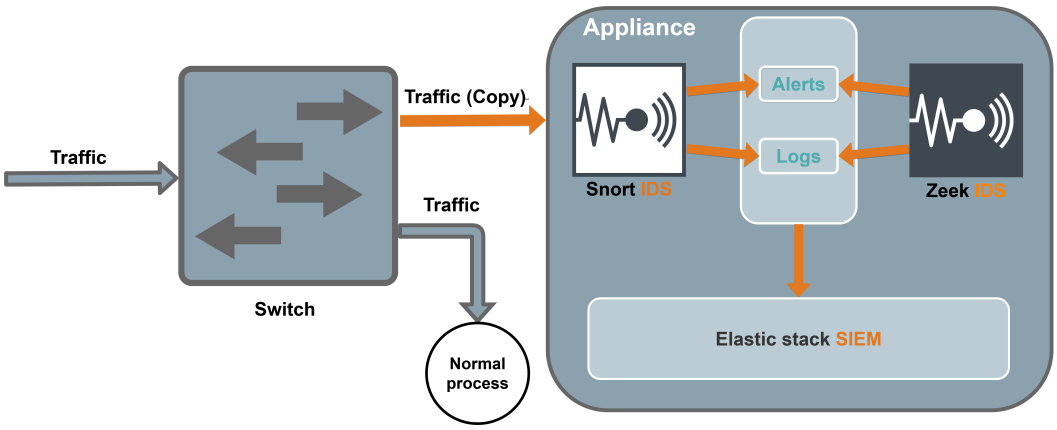

Figure 4.3: ICS-Security-Appliance intrusion detection and monitoring architecture revisited

4.6 Attacker Model

Attackers usually follow an attack vector in order to compromise networks and systems. Attack vectors are methods or strategies of intrusion used by attackers to materialize their actions into a potential threat [56].

In order to compromise the industrial control network (which shelters every industrial process) it is expected that attackers have been able to:

- Establish a foothold on the corporate network remotely, by carrying out a phishing attack
- Obtain all the information regarding the presence and access points to the industrial control network from the corporate network
- Obtained schematics and inventory information regarding the industrial control network and its underlying devices

To achieve intrusion in the corporate network and pivot to the industrial control network, it is expected that attackers have access to:

- Malicious files to send as attachment in phishing attacks to gain control of the workplace computer that opens it
- Penetration testing and attack frameworks like Metasploit and Ettercap (Section 2.5.4 and Section 2.5.5)
- Knowledge of the fundamentals of ICS operations
- Knowledge of the fundamentals of protocols like DNP3, Modbus, S7Comm and IEC 61850 MMS
- Default access credentials for control devices

4.7 Baselining

With regards to baselining traffic, some changes were added to the default implementation of Bropy. A new option was added to the main menu of Bropy, that allows the user to generate several baseline files, based on different criteria. For each of these baseline files to be generated, a script was implemented that reads Zeek logs and other files of interest, organizes the data and writes to a corresponding tab separated values file.

These baseline files are later accessed by the ICS Appliance's Zeek implementation, to verify if traffic is within baseline, with the help of Zeek's implemented policies/detection scripts.

To better understand the importance of this Bropy implementation and what baseline files are generated, Table 4.1 depicts a representation of these files, their header fields and a small description of what the file is responsible for. The left column portrays the tab separated fields to be expected in the file, while the right column contains a small description of the role the baseline file has within detection notions.

Endpoints	
Dest IP / Dest Port / Proto / Source IPs	All host communications; Used to check if a source host is allowed to talk to the destination host
Route Table	
Source IP / Source MAC	Used to ensure ARP messages reply with correct IP:MAC (for MITM)
Control Hosts	
Source IP / Dest IP / Dest Port / Proto / Instructions	All hosts that issue instructions; Used to check if an instruction is usually sent from a src.host to a dest.host on a dest.port
Devices	
IP / MLFB / Device	All devices to which an MLFB was seen; Used to ensure that there are no new devices on the network; Add verbosity to alerts
DNP3	
Dest IP / Dest Port / Function / Source IPs	All host communications to DNP3 slaves; Used to check if a src.host is allowed to talk to the DNP3 slave and execute a particular function code
Modbus	
Dest IP / Dest Port / Function / Sources IPs	All host communications to Modbus slaves; Used to check if a src.host is allowed to talk to the Modbus slave and execute a particular function code
Credentials	
Vendor / Device / Credentials	Known default credentials for devices; Used to check if a login to a device service from specific vendor contains a defaul credential

Table 4.1: Baseline description for every baseline file

A top down view of baseline implementation is as follows:

- Endpoints
  - Zeek registers every connection in its default connection logs
  - A default Bropy python script reads connection logs, extracts **Dest IP**; **Dest Port**; **Protocol** and **Source IP**
  - For each entry, Bropy then writes a line to a file, that portrays every **Source IP** that communicated with **Dest IP** on **Dest Port** through **Protocol**, all separated by tabs.
- Route Table
  - Zeek registers every connection in its default connection logs
  - an implemented Bropy python script reads connection logs, extracts every **Source IP** and its corresponding **Mac address**

  - For each entry, Bropy then writes a line to a file, that protrays the **Source IP** and it's corresponding **Mac address**, separated by tabs.

- Control Hosts

  - Zeek registers every connection and corresponding issued command, for some S7Comm and IEC-61850-MMS instructions, on an implemented control_hosts log (**Source IP**; **Destination IP**; **Port**; **Protocol** and **Command**);
    - For the S7Comm protocol, the implemented Zeek script searches for "Stop service", "Program logic Upload" and "Program logic Download" instructions (explained in Section 2.1.4), via pattern matching, on the TCP packets.
    - For the IEC-61850-MMS protocol, the implemented Zeek script searches for any characters coming after TPKT packet header (0x03 0x00 0x?? 0x??) and ISO 8073 COTP packet header (0x02 0xf0 0x80) and checks them against a constants table reflecting the PDU types (both explained in Section 2.1.4).
  - An implemented Bropy python script that reads Zeek's implemented control_hosts log to extract **Source IP**, **Destination IP**, **Protocol** and the issued **Command**;
  - For each entry, Bropy then writes a line to a file, that portrays the **Source IP**, the **Destination IP**, the **Port**, the **Protocol** used and all the **Commands** that were issued from the **Source IP** to the **Destination IP**, all separated by tabs.

- Devices

  - Zeek registers every connection to which an MLFB was seen in its implemented devices log (**Source IP**; **Destination IP**; and **MLFB**)
  - A manually created file, containing the name of all equipment and their corresponding MLFB
  - An implemented Bropy python script that reads the manually created equipment file and Zeek's devices log to outline the presence of the actual device on the network
  - For each entry, Bropy then writes a line to a file, that portrays the **IP**, **MLFB** and the name of the Device, all separated by tabs.

- DNP3

  - Zeek registers every DNP3 connection in its default DNP3 log
  - An implemented Bropy python script reads DNP3 logs, extracts **Dest IP**; **Dest Port**; **DNP3 function code** and **Source IP**
  - For each entry, Bropy then writes a line to a file, that portrays every **Source IP** that communicated with **Dest IP** on **Dest Port** using a specific **DNP3 funcion code**, all separated by tabs.

- Modbus
  - Zeek registers every Modbus connection in its default Modbus log
  - An implemented Bropy python script reads Modbus logs, extracts **Dest IP**; **Dest Port**; **Modbus function code** and **Source IP**
  - For each entry, Bropy then writes a line to a file, that portrays every **Source IP** that communicated with **Dest IP** on **Dest Port** using a specific **Modbus funcion code**, all separated by tabs.
- Credentials
  - Manually created file with **Vendor**, **Device** and **Credentials** fields, separated by tabs

4.8 TTPs and Detection Use Cases

This section encompasses a list of tactics, techniques, and procedures (TTPS) carried out by threat actors along with potential use cases for the detection of these TTPs that are important from an ICS perspective when pivoting from an IT environment. Detailing how it could be possible to detect these threats by providing a general guideline to keep in mind when trying to defend these systems. An example, for some of these use cases, has been implemented in the developed ICS appliance as will be discussed further along in this section.

Table 4.2 depicts example use-cases for detection and their corresponding TTP category, by assessing the *ATT&CK for Industrial Control Systems* from the Mitre organization [45]. Some of these examples are not included in Mitre's ICS Attack Matrix, thus extrapolation was taken into consideration to help conjugate them with potentially related TTP categories

Use Cases	TTP
Communication Channel	Command and Control
Data Exfiltration	Command and Control
Enumeration scripts	Discovery
Man In The Middle	Execution
Default Credentials	Lateral Movement
Program logic Upload	Collection
Program logic Download	Persistence, Inhibit Response Function, Impair Process Control
Service Stop	Impair Process Control
Segment Smack	Impair Process Control, Inhibit Response Function
System Firmware	Persistence, Inhibit Response Function
Activate Firmware Update Mode	Inhibit Response Function
Force Listen Only	Inhibit Response Function
Disable Unsolicited Messages	Inhibit Response Function

Table 4.2: Use cases and their corresponding TTP

4.8.1 Unusual Endpoints

On networks where traffic is more deterministic, it is easier to establish normal behavior. Once we know how machines behave and relate to each other we can expect traffic to remain relatively constant. The same IP's always communicate with each other on the same ports. If a security analyst is aware of the network requirements, then it is relatively effortless to point out deviations from the expected behavior. This is particularly useful for industrial control networks, as machines and devices seldom change their behaviors.

We can consider an endpoint, the destination IP, destination port and protocol used from a connection. By singling out any traffic that does not conform with an expected connection from a particular IP to a particular endpoint, it is possible to highlight a potential intrusion.

Use-Case

- Detect a communication that deviates from an established traffic baseline
  - Establish a baseline of communication (Who talks with whom on which ports and through which protocols)
  - Employ detection mechanisms that verify if traffic is within baseline

Requirements

- Have any machine communicating on testing environment
- Wait for connection logs to be created
- Execute Bropy baselining tool with extended scripts
- Force different communication (Add a new device, for example)
- Wait for alerts to be generated

Implementation

For this implementation, the default baseline scripts (**bropy conparse**) and default Zeek detection script (**baselinereport zeek script**) taken from Bropy's defaut implementation, were taken advantage of. Some alterations were done on the baselinereport Zeek script to reflect a different description of this attack's occurence - "Endpoint_Baseline_Deviation" in place of the original "TrafficbaselineException".

- Hosts generate traffic

- Zeek registers traffic in conn logs
- Bropy reads conn logs and generates a file with **dest.ip**, **dest.port**, **protocol** and all **src.ips** that communicate with **dest.ip**.
- Zeek listens to new traffic and for every new connection established, compares if **src.ip** communicates with **dest.ip** on **dest.port** through **protocol** from the baseline file

In Section 5.2.1 we can see this use case in action, as well as it's assessment.

4.8.2 Communication Channels and Data Exfiltration

Some systems, including those pertaining to ICS networks, can be compromised and allow for attackers to remotely manipulate process control and industrial data. In order to remain undetected, attackers often try to leverage commonly used ports and protocols in an effort to mimic expected network traffic to avoid suspicion and establish a communication channel/tunnel between the attacker and a compromised machine on the network. These communication channels can also be used to exfiltrate data out of the network to later be used by the attacker (User credentials, Architecture design, etc) . **Command and Control** represent the techniques performed by attackers to establish a communication channel between a remote server and compromised systems on a network in order to issue commands to these systems and exfiltrate data.

Usually, a victim downloads an infected document, from a nefarious email, that installs either a server or a client software on the victim machine. This server can then be accessed by an attacker in order to pull data and send commands or the client software can communicate directly to an attacker's server outside the network, for the same purpose. Later on, the attacker can leverage this advantageous position on the victim's network to establish persistence on new hosts, until the attacker can pivot to the industrial control network, on which he may then be able to issue commands to devices and disrupt industrial process.

DNS: According to Greg Farnham, in [19], for DNS tunneling to succeed, data is often included in large quantities into requests and responses which result in requests having labels of up to 63 characters and names of up to 255 characters. Guy, J. [23] also proposes to look at hostname requests longer than 52 characters. The presence of an uncommon record type, can also be of interest. Pietraszek, T. and Mertens [59] argue that looking for records that are not commonly used, for example the TXT record could be another indication of an attack. "DNS tunnelling is a common way to establish connections with remote systems. It is often based on "TXT" records used to deliver the encoded payload" [59]. DNS is universally used in every infrastructure to connect public and private networks to the internet. It is less likely to be monitored or filtered as it is a big necessity.

ICMP: Through the ICMP protocol it may also be possible to establish a covert channel that can sometimes be overlooked by firewalls, as the ICMP protocol is a simple protocol often used for network troubleshooting. If ICMP messagens are not blocked by a firewall, an attacker could leverage this protocol, not only for network scans (further discussed in Section 4.8.3), but also to establish a covert channel between the attacking host and a victim host in order to send commands and exfiltrate data to and from the victim. Furthermore, ICMP echo reply data is exacly the same as ICMP echo request data, thus if reply data differs from request data it may be because data is being sent through ICMP packets (data is different on every request and reply). A very common way to leverage ICMP to establish a Command and Control server is through the use of PowerShell scripts - PowerShell is a Microsoft framework with its own scrpting language and command-line shell, used for automation of tasks and configuration management.

Use-Case

- Detect the presence of Command and Control activity by assessing the existence of covert channels and data exfiltration through DNS
  - Look at DNS requests
  - Assess if TXT record type is being used
  - Verify the ammount of characters in DNS Domain name
  - Verify the ammount of times a long DNS Domain name was accessed and if the characters remain the same (otherwise it could be an indication of data being exfiltrated)
- Detect the presence of Command and Control activity by assessing the existence of covert channels and data exfiltration through ICMP
  - Look into ICMP packets
  - Verify presence of PowerShell activity
  - Verify if ICMP echo request data and ICMP echo reply data contents match

Requirements

- DNS
  - Execute a DNS server on attacking computer to wait for connection from the victim
  - Execute a DNS client script on a victim computer that communicates with attacker
  - Have victim and attacker communicating
  - Understand how to parse DNS messages
  - Obtain DNS query names and record type

  - Wait for alerts to be generated

- ICMP
  - Execute ICMP server on attacking computer to wait for connection from the victim
  - Execute ICMP PowerShell client script on victim computer that communicates with attacker
  - Have victim and attacker (KALI) communicating
  - Understand how to parse ICMP messages
  - Obtain ICMP data echo
  - Wait for alerts to be generated

Implementation

- DNS
  - DNS reverse shell script executes on victim machine
  - A DNS channel is established between victim machine and attacker machine
  - DNS traffic is being generated between both the victim machine and attacking machine
  - Zeek listens to dns traffic, and for ever DNS request:
    * If query length char threshold is over 52 chars, flag it as an attack and alert (tunnel established)
    * If the amount of times query length is over 52, is more than a specified threshold, flag it as an attack and alert (data exfiltration)
    * If query is a TXT record (suspicious), flag it as an attack and alert (control or data exfiltration)

- ICMP
  - Powershell script executes on victim machine
  - Sends icmp packets to attacker machine (KALI)
  - Zeek listens to icmp traffic
    * if PowerShell related keywords are detected (for example ”PowerShell”, ”Windows”, ”Microsoft Corporation”, ”PS C:
      Users
      .*¿”; ”Windows PowerShell running as user .*”, among others) flag it as an attack and alert (tunnel established)
    * if echo reply data is different than echo request data, flag it as an attack and alert. (control or data exfiltration)

In Section 5.2.2 we can see the use case in action and its generated alerts.

4.8.3 Discovery Techniques

Remote system discovery and network service scanning are processes for identifying the presence of hosts and services on the network. An attacker may attempt to gain information about the target network via network enumeration and discovery techniques such as port scanning, which produce information relating to existing devices, running services and specifications. These techniques, although useful from a network management standpoint, can be used for nefarious means. An attacker can use this information to understand what devices comprise the network and even select possible exploits for a particular version of a device if there are any known vulnerabilities for it.

Use Case

- Verify if a host is trying to sniff the network for Siemens Simatic PLC devices
- Verify if a host is trying to sniff the network for Communications Processors on Simatic devices
- Verify if a host is trying to sniff the network for Scalance devices
- Verify if a host is trying conduct a port scan

Requirements

- Have S7-1200, S7-1500 and/or S7-300 and a Scalance (CPs not available, but there are pcaps) communicating on the network
- Wait for logs to be created
- Execute bropy
- Execute Nmap tool from attacking machine:
  - Siemens s7-PLCs (Siemens-SIMATIC-PLC-S7.nse and s7-enumerate.nse) nmap scripts
  - Communications Processor for Siemens devices (Siemens-CommunicationsProcessor.nse nmap scrip)
  - Siemens SCALANCE switch modules (Siemens-Scalance-module.nse nmap script)
  - Port Scan
- Wait for alerts to be generated
  - Siemens Simatic S7 PLC
  - Comunications Processor
  - Scalance
  - Port Scan

Implementation

Through Zeek

- Hosts generate traffic
- Zeek listens to traffic and:
  - For every SNMP report, verifies if it corresponds to the "**1.3.6.1.2.1.1.1**" OID (System Description), if so, flags it as an attack and alerts
  - For every HTTP request, verifies, trough pattern matching, if the request is an attempt at recognizing a Communications Processor (URI containing **"/Portal0000.htm"** and **"/__Additional"**) or a Simatic device (URI containing **"/docs/cplugError.html/"** and **"/Portal/Portal.mwsl"**), if so, flags it as an attack and alerts
  - By leveraging Zeek's default Scan policy script, detecting port scans is possible.

Through Snort

- Deploy DigitalBond's Snort rule on Snort software within ICS Appliance
- Have machines communicating
- Execute nmap script s7-enumerate.nse on attacking machine to enumerate s7 simatic devices
- Snort detects rule contents inside UDP payload, sent from attacker
- Wait for alerts to be generated

In Section 5.2.3 we can see the use case in action and its generated alerts.

4.8.4 New Executable files

According to Slowik [62], ICS security must include detection of different strategies that can help identify, for example, new executable files entering into critical environments. This is true, particularly, when pivoting from the IT network to the industrial control network.

It is very dangerous to have executable files being transferred onto a network. They may contain malicious code that would, most certainly, be used to compromise network components.

Use Case

- Detect an executable file being transferred to a protected network

  - Look at traffic
  - Assess traffic to find a file that is being transferred
  - Extract file type
  - Verify if file is an executable

Requirements

- Have attacking machine (Kali) and Engineering Workstation communicating on the network.
- Transfer an executable file from the attacking machine to the engineering workstation (representing an executable file coming into a protected network)
- Understand how a file transfer is conducted on the network
- Understand how to extract the file type
- Wait for alerts to be generated

Implementation

By leveraging Zeek's logging, file analysis framework, and parsing capabilities it was possible to test MIME types (detailed in Section 2.5.6) of files being transferred through the network, against a list of known MIME types related to executable files. If these MIME-types where seen by Zeek, and they are usually not seen on the network or, at the very least, on the specific part of the network they were detected in, an alert should be generated.

- Hosts generate traffic
- Zeek listens to traffic and every time he sniffs a file on the network, checks the file's mime type to figure out if it is an executable (mime types represent the format of a file):
  - /application/vnd.microsoft.portable-executable
  - /application/octet-stream
  - /application/x-dosexec
  - /application/x-msdownload

In Section 5.2.4 we can see the use case in action and its assessment.

4.8.5 Man in the Middle

Once an attacker gains privileged access to a network and knows which devices communicate, he may seek to intercept traffic to and from devices. Through a Man in the Middle (MITM) attack, an attacker can position himself between two communicating hosts while acting as each of them - Hosts do not realise they are not talking to each other, but with a malicious actor instead. If a MITM attack is established, an attacker can manipulate traffic however he wishes. One of the most common ways to accomplish this attack is by performing ARP poisoning (Section 2.5.3).

Use Case

- Detect ARP poisoning attacks that may infer a MITM
  - Verify if there was the same mac for different IPs
  - Verify if the same IP had different mac
  - Outline the similarities to find culprit

Requirements

- Have machines communicating on testing environment
- Wait for connection logs to be created
- Execute Bropy baselining tool with extended scripts
- Use Ettercap (Section 2.5.5) to establish a MITM position between two hosts
- Wait for alerts to be generatedd

Implementation

- Hosts generate traffic
- Zeek registers traffic in conn logs
- Bropy reads conn logs and generates baseline file with **src.ip / src.mac**
- Zeek listens to new traffic (every ARP reply) and compares if **src.ip** : **src.mac** is correct based on the previously generated baseline file.

4.8.6 Default Credentials

In ICS the default credentials for accessing devices and it's services are rarely changed after deployment, because the focus is on operation rather than security. By detecting that default login credentials are being used for login purposes we have reason to suspect the presence of an attacker in our network as it is common for attackers to try and gain access by exploiting the fact that credentials may not have been changed since first implemented.

Use Case

- Detect an attempted login using default credentials
  - Obtain the payload from traffic
  - Verify, if possible, if the payload consists of a login and which credentials it was using
  - Compare those credentials to known ICS device's default credentials
- Detect several login attempts using default credentials from same host
  - Obtain the payload from traffic
  - Verify, if possible, if the payload consists of a login and which credentials it was using
  - Compare those credentials to known ICS device's default credentials
  - Register number of occurrences of this nature until it crosses a defined threshold

Requirements

- Hosts generate traffic
- Wait for logs to be created

Implementation

- Have an S7-300 device and Engineering Workstation communicating on the network
- **Zeek** registers traffic in devices log (ips and mlfbs);
- **Bropy** reads devices log and a manually created device csv file (containing device type and its corresponding **MLFB**) and generates a baseline file with **src.ip**, **device** and **MLFB** (baseline_devices.data);
- Zeek listens to new traffic and for every tcp packet, looks at the payload and looks for the occurrence of **/Login=.*&Password=.*/** (common format of login to the s7 devices webserver), and compares it to a manual baseline file that contains known default passwords for

known devices, as well as the baseline_devices.data to understand if the **src.ip** is leveraging known default credentials to gain access to dest.ip which can, in fact, be the corresponding device to those credentials.

  - Also checks for the occurrence of several login attempts across several devices to check for password spraying (using Zeek's Summary Statistics framework)

4.8.7 Program Logic Download

Control devices, such as the PLCs run a logic program that dictates how field devices are supposed to operate. These programs are downloaded from an engineering workstation or control center onto the device itself.

When infiltrated on a network, attackers may take advantage of this to perform a program download onto devices, allowing the attacker to implement custom logic. Malicious PLC programs may be used to disrupt physical processes or to establish attacker persistence. The act of a program download will cause the PLC to enter a STOP operation state, which may prevent response functions from operating correctly.

The Stuxnet malware [18], for example, injects PLCS with different logic through code blocks and data blocks by downloading them to the device's firmware, effectively altering it's behavior.

Use Case

- Detect a download instruction coming from a host along with syslog info from the PLC, reflecting a shutdown.
  - Verify TCP payload for a *Program Logic Download* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
- Program download out of work hours
  - Verify TCP payload for a *Program Logic Download* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if time was within work hours
- Program download from a host not whitelisted to do so
  - Verify TCP payload for a *Program Logic Download* instruction

  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if host is allowed to perform *Program Logic Download* instructions

Requirements

- Understand what a download instruction looks like
- Obtain syslog info to correlate both events and decide if the *Program Logic Download* was successful
- Obtain a list of hosts (engineering Workstations) allowed to download programs
- The general work hour for that specific host

Implementation

- Hosts generate traffic
- Zeek registers traffic in control_host log (ips and command instructions)
- Bropy reads control_hosts log and generates file with src.ip and command instruction (baseline_endpoints.data)
- Zeek listens to new traffic and for every TCP packet, looks at the payload and verifies if it is an S7Comm *Program Logic Download* instruction (and which data blocks are being downloaded based on pre-defined constants in consts_S7blocks.zeek), and compares it with the baseline file (baseline_endpoints.data) to understand if the **src.ip** responsible for issuing that **instruction**, can, in fact, do it.

4.8.8 Program Logic Upload

It is very common for attackers to execute the upload of a PLC's program logic in order to gather information about industrial process. By uploading information from the PLC, attackers can understand and study the logic behind the industrial process in order to formulate a specialized attack.

Use Case

- Detect an upload instruction coming from a host along with syslog info from the PLC, reflecting a shutdown.
  - Verify TCP payload for a *Program Logic Upload* instruction

  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)

- Program logic Upload out of work hours
  - Verify TCP payload for a *Program Logic Upload* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if time was within work hours
- Program download from a host not whitelisted to do so
  - Verify TCP payload for a *Program Logic Upload* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if host is allowed to perform *Program Logic Upload* instructions

Requirements

- Understand what an upload instruction looks like
- Obtain syslog info to correlate both events and decide if the *Program Logic Upload* was successful
- Obtain a list of hosts (engineering Workstations) allowed to upload programs
- The general work hour for that specific host

Implementation

Much like the **Program Logic Download** the implementation for this use-case is exactly the same:

- Hosts generate traffic
- Zeek registers traffic in control_host log (ips and command instructions)
- Bropy reads control_hosts log and generates file with src.ip and command instruction (baseline_endpoints.data)
- Zeek listens to new traffic and for every TCP packet, looks at the payload and verifies if it is an S7Comm *Program Logic Upload* instruction (and which data blocks are being uploaded based on pre-defined constants in consts_S7blocks.zeek), and compares it with the baseline file (baseline_endpoints.data) to understand if the **src.ip** responsible for issuing that **instruction**, can, in fact, do it.

- Zeek also checks the timestamp of the occurrence to verify if it is within workhours (hard-coded value from 9am to 5pm)

4.8.9 Service Stop

An attacker may be responsible for stopping or disabling services on a device, which would force them to be unavailable, prevent response to an incident or even allow the attacker to further compromise the industrial environment.

Use Case

- Detect a *Stop* instruction coming from a host to a control device, along with a syslog message from the device reflecting a shutdown
  - Verify TCP payload for a *Stop Service* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
- *Stop* instruction out of workhours
  - Verify TCP payload for a *Stop Service* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if time was within work hours
- Detect a *Stop* instruction from a host not whitelisted to do so
  - Verify TCP payload for a *Stop Service* instruction
  - Verify syslog info for shutdown indicator (To understand if the attack was successful or unsuccessful)
  - Verify if host is allowed to perform *Stop Service* instructions in baseline

Requirements

- Understand what a *Stop Service* instruction looks like
- Obtain syslog info to correlate both events and decide if the *Stop Service* was successful
- Obtain a list of hosts (engineering workstations) allowed to issue a *Stop Service* instruction
- The general work hour for that specific host

Implementation

This detection use-case was implemented for both the S7Comm protocol, a *Service Stop* instruction, and for the DNP3 protocol, a *Stop application* function code. The implementation for each, is as follows:

- S7comm
  - Hosts generate traffic
  - Zeek registers traffic in control_host log (ips, command instructions)
  - Bropy reads control_hosts log and generates file with src.ip and command instruction (baseline_endpoints.data)
  - Zeek listens to new traffic and for every tcp packet, looks at the payload and verifies if it is an s7comm stop instruction, and compares it with the baseline file (baseline_endpoints.data) to understand if the src.ip responsible for issuing that instruction, can, in fact, do it.
- DNP3
  - Hosts generate traffic
  - Zeek checks for function codes associated with DNP3 and registers that traffic in dnp3 log (ips, function codes)
  - Bropy reads dnp3 log and generates file with dst.ip, function code and every src.ip that issues that function to dst.ip (baseline_dnp3.data)
  - Zeek listens to new traffic and for every dnp3 message, looks at the header and verifies if the function code instruction corresponds to a *STOP_APPL* (0x12), if so, flag it as an attack and alert

4.8.10 Segment Smack

Segment Smack (CVE-2018-5390), forces 4.9+ kernel versions of Linux to make expensive calls to some CPU intensive functions (tcp_collapse_ofo_queue() and tcp_prune_ofo_queue()), for every packet sent. This is achieved by bombarding the target machine with TCP keep-alive packets (PSH; ACK). It may not be a specific exploit for ICS components, but its presence in the network is enough to sever the connection between the PLC and the HMI.

Use Case

- Detect the Segment Smack attack on the network
  - Verify TCP connections for a great amount of TCP keep-alive packets.

Requirements

- Understand how the segment smack works
- Apply an IDS rule that verifies TCP keep-alive packets

Implementation

In order to detect this attack, a simple Snort rule that would detect if, for a period of 60 seconds, at least 10 packets with the flags PSH; ACK were sent to the PLC on port 102, was deployed on the ICS Appliance.

```
alert tcp any any -> $HOME_NET 102 (flags: PA; threshold:type both,
track by_src, count 10, seconds 60; msg:"Possible Segment Smack exploit
through several PUSH ACK packets"; reference:cve, CVE-2018-5390; priority:1;
sid:11111112)
```

4.8.11 System firmware

Updating device firmware is fundamental to improve operations and security and it is, usually, openly available for download. Updating firmware (as important as it may be) is usually postponed, as it can impair industrial process. An attacker may leverage this to tamper with a firmware update file and then update the device with a malicious firmware, while also inducing a temporary halt in process.

Use-Case

- Detect a malicious firmware update file
  - By gaining access to an available firmware file from a vendor's website
  - Tamper with the firmware file, effectively turning it malicious
  - Upload the file to a device in order to compromise it;
  - Obtain the file's checksum/hash while its being uploaded through the network;
  - Compare the checksum/hash obtained with one known as benign for that known firmware update.

Requirements

- Understand how a firmware update instruction or firmware file upload looks like from a network perspective;

- how to tamper with said firmware file (through reverse engineering)
- how to obtain an hash of the firmware on the network.

Implementation

[Implementing this use case was not possible]

- Hosts generate traffic
- Zeek listens to traffic and every time he sniffs a file on the network, verifies if it is a known firmware file
- Zeek calculates the file's hash value
- Then compares the hash value to the known file's benign hash value. If they differ, flag it has malicious and alert

In Section 5.2.7, an assessment on this use-case and its lack of implementation.

4.8.12 Activate Firmware Update Mode

As a device enters a firmware update mode, it may halt its process and functions. Having a device remain in firmware update mode by not providing any firmware files to update, is of special importance to threat actors. By forcing a device to enter a firmware update mode, and leaving it as is, an attacker can cause a DoS condition.

According to Joe Slowik from Dragos Inc [61], The Industroyer (Crashoverride) SIPROTEC DoS module exploits the CVE-2015-5374 vulnerability which places the victim device into a "firmware update mode" by sending specifically crafted packets to port 50000 of the target device via UDP. This specifically crafted packet, contains **0x11 49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 28 9E** 18 byte content in its payload. The target device will not respond to any instructions up until the time when it is rebooted manually.

Use Case

- Detect SIPROTEC DoS attack
- Verify the ocurrence of 18 byte payload: 0x11 49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 28 9E in UDP packet to port 50000.

Requirements

- Snort Rule from SourceFire Inc which detects a SIPROTEC DoS attack;
- Zeek policy that verifies the presence of 0x11 49 00 00 00 00 00 00 00 00 00 00 00 00 00 00 28 9E within UDP packets to port 50000.

Implementation

To detect this specific attack that induces an *active firmware update mode*, a Snort rule, created and certified by Sourcefire, was deployed on the ICS-Security-Appliance's Snort IDS.

- Deploy SourceFire's Snort rule on Snort software within ICS Appliance
- Have machines communicating
- Start metasploit attacking framework on attacking machine
- Issue SPIROTEC DoS module to place the victim device into "firmware update" mode.
- Snort detects rule contents inside UDP payload, sent from attacker
- Wait for alerts to be generated

4.8.13 Force Listen Only

As explained before, in Section 2.1.4, the Modbus protocol works through function codes that represent commands being sent to Modbus devices. The *Diagnostics* function code (x08), contains several other sub function codes, one of which represent a *Force Listen Only Mode* (0x08 - 0x04). When a Modbus device is sent a *Force Listen Only Mode* instruction, it will become inactive, not responding to other commands or sending any other responses. This command is often used to trigger a Denial of Service condition on these devices. To restore functionality, the devices have to be restarted.

Use-Case

- Detect a *Force Listen Only Mode* instruction from an attacking machine to a Modbus device
  - Find Modbus payload on the wire
  - Verify presence of funcion codes 0x08 and 0x04 (content: `|08 00 04|`)

Requirements

- Have S7-1200, S7-1500 and/or S7-300 and Engineering Workstation communicating on the network with a Modbus program (if not possible use a PCAP example or establish a Modbus server and a Modubs client on separate machines)
- Wait for traffic logs to be created
- Execute Bropy to baseline
- Issue a Modbus *Force Listen Only Mode* (0x08 – 0x04)
- Understand how to parse Modbus payload to verify existance of *Force Listen Only Mode* command

Implementation

Originally, the implementation is as follows:

- Hosts generate traffic
- Zeek checks for function codes associated with Modbus and registers that traffic in Modbus log (ips, function codes)
- Bropy reads Modbus log and generates file with **dst.ip**, **function code** and every **src.ip** that issues that function to **dst.ip** (baseline_modbus.data)
- Zeek listens to new traffic and for every Modbus message, looks at the header and verifies if the function code represents a *Diagnostics* (0x08) instruction, which is the main function code for a *Force Listen Only* as explained before.
- If Zeek finds, indeed, the presence of a *Diagnostics* instruction, flags traffic as potentially malicious and alerts. Even if this instruction is base lined, the alert still triggers, as it should not be too common to issue this command.

As will be explained in Section 5.2.8, the previous implementation is not the foremost way of detecting the *Force Listen Only Mode* instructions. To that end, a Snort rule was deployed on the ICS Appliance's Snort implementation.

```
alert tcp any any -> $HOME_NET 502 (flow:to_server; content:"|00
00|"; offset:2; depth:2; content:"|08 00 04|"; offset:7; depth:3; msg:
"Modbus - Force Listen Only Mode"; classtype:attempted-dos; priority:1;
sid:11111111;)
```

Through this Snort rule, it is possible to inspect the TCP packet for a sub-function within the Modbus payload. This alternative implementation goes as follows:

- Hosts generate traffic
- Snort parses every TCP packet going to the configured protected network on port 502 (Modbus port)
- Snort Inspects the packet for the content |00 00| (depth two) on an offset of two starting at the beginning of the Modbus payload until it can reach the content |08 00 04| (depth three) on an offset of seven.

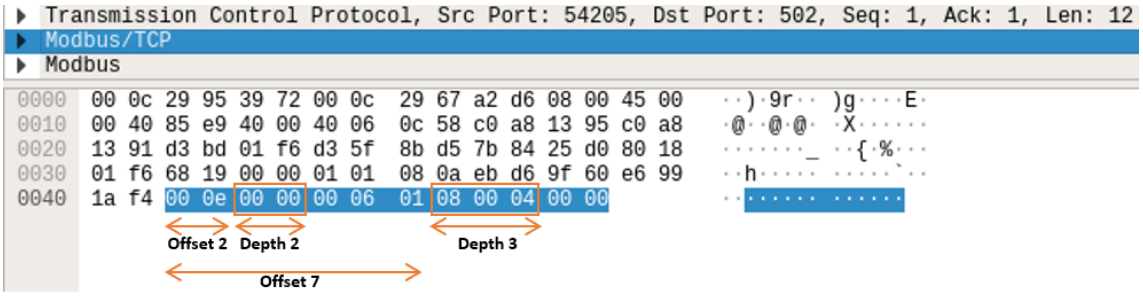

Figure 4.4: Representation of the Modbus payload detailing a Force Listen Only Mode instruction on Wireshark

In Section 5.2.8 we can see the use case in action and its assessment.

4.8.14 Disable Unsolicited Messages

Through unsolicited messages, remote DNP3 slave devices can transmit event messages (i.e warnings, errors, specific readings, among others) to a master station without the master needing to poll that data on command. This information is imperative to elicit a response from the master station in order to take immediate action and deal with potential problems or changes in production values [11].

If a DNP3 master is to send a *Disable Unsolicited Messages* command to a DNP3 slave device - function code 21 (0x15), this instruction, will cause the target device to stop sending unsolicited messages. Disabling these messages can cause loss of visibility from the DNP3 master's perspective. If a DNP3 master cannot respond and deal with an incorrect state of production, more disastrous outcomes could come to fruition.

Use-Case

- Detect a *Disable Unsolicited Messages* instruction from an attacking machine to a DNP3 device
  - Find DNP3 payload on the wire
  - Verify presence of function code 21 (0x15) (content: `|15|`)

Requirements

- Have S7-1200, S7-1500 and/or S7-300 and Engineering Workstation communicating on the network with a Modbus program (if not possible use a PCAP example or establish a Modbus server and a Modubs client on separate machines)
- Wait for traffic logs to be created
- Execute Bropy to baseline
- Issue a DNP3 *Disable Unsolicited Messages* (0x15)
- Understand how to parse DNP3 payload to verify existence of *Disable Unsolicited Messages* command

Implementation

- Have S7-1200, S7-1500 and/or S7-300 and Engineering Workstation generating traffic on the network with a DNP3 program (if not possible use a PCAP example)
- Zeek checks for function codes associated with DNP3 and registers that traffic in a DNP3 log (ips, function codes)
- Bropy reads dnp3 log and generates file with **dst.ip**, function code and every src.ip that issues that function to **dst.ip** (baseline_dnp3.data)
- Zeek listens to new traffic and for every dnp3 message, looks at the header and verifies if the function code instruction corresponds to a *DISABLE_UNSOLICITED* (0x15), if so, flag it as an attack and alert

4.8.15 Unusual Control Instructions

The importance of establishing normal behavior has already been discussed, thus we can assume that establishing expected control instructions to and from control devices is imperative to gain

visibility into malicious behavior from an industrial process perspective. Understand which devices and hosts are issuing commands/instructions, and what are those commands/instructions. If an attacker gains control of an host responsible for controlling an industrial device, it would be possible to issue malicious instructions to the device in order to disrupt production.

Use-Case

- Hosts generate traffic
- Issue instructions from a controlling host to a device
- Baseline those communications
- Issue a new instruction not present on the baseline
- Find the payload on the wire
- Parse the payload to obtain instruction and compare it to baseline

Requirements

- Understand how different protocols work
- Understand how to baseline instructions
- Develop or use a mechanism that parses the payload to find an instruction

Implementation

The SCADA protocols focused on for this example were DNP3, Modbus, S7Comm and IEC-61850-MMS. Zeek's protocol analyzers for DNP3 and Modbus were useful, which allowed for easier payload parsing from these protocols. For the S7Comm protocol, instructions were limited to *Service Stop*, *Program Logic Upload* and *Program Logic Download* via pattern matching on TCP packets. For the IEC-61850-MMS protocol (Section 2.1.4), the focus was on MMS PDU instructions, by establishing constants that represent these instructions and verifying their presence in TCP packets, also via pattern matching.

The DNP3 and Modbus implementation is as follows:

- Have S7-1200, S7-1500 and/or S7-300 and Engineering Workstation communicating on the network with a Modubus or DNP3 program
- Zeek checks for function codes (instructions) associated with either DNP3 or Modbus, and registers that traffic in DNP3 and Modbus logs (ips, function codes).

- Bropy reads DNP3, Modbus logs and generates files from both protocols, with **dst.ip**, **instruction** and every **src.ip** that issues that function to **dst.ip**.
- Zeek listens to new traffic and for every modbus message or dnp3 message, looks at the header and verifies if the function code instruction is baselined, comparing it with the baseline files to understand if the **src.ip** responsible for issuing that instruction to **dst.ip**, can, in fact, do it.

The S7Comm and IEC-61850-MMS implementation is as follows:

- Have S7-1200, S7-1500 and/or S7-300 and Engineering Workstation communicating on the network with a S7Comm (IEC-61850-MMS had to be simulated with packet captures)
- Zeek looks at the TCP payload and, via pattern matching, checks for instructions associated with either S7Comm or IEC-61850-MMS, and registers that traffic in a *Control Hosts* log with **src.ip**, **src.port**, **dst.ip**, **dst.port protocol** and the **instruction**.
- Bropy reads *Control Hosts* log and generates a file , with **src.ip**, **dst.ip**, and which **instructions** are issued from **src.ip** to **dst.ip**.
- Zeek listens to new traffic and for every TCP packet, looks at the payload, searching for instructions via pattern matching and verifies if the instruction is baselined, comparing it with the baseline files to understand if the **src.ip** responsible for issuing that instruction to **dst.ip**, can, in fact, do it

In Section 5.2.9 and Section 5.2.10 we can see the use-case in action for both S7Comm and IEC 61850 MMS protocols respectively, as these are of significant importance to discuss.

4.8.16 Other ICS attacks

This use-case exists to add further basis for using the Snort-IDS tool in the ICS-Security-Appliance

Use Case

- Detect a DATAC RealWin SCADA Server Buffer Overflow
  - Attacker executes arbitrary code by sending crafted packet with the content ”FC_INFOTAG/SET_CONTROL” to DATAC RealWin Server 2.0 (Build 6.0.10.37) [42];
  - Deploy a Snort rule that detects this specially crafted packet
- Detect a Sielco Sistemi Winlog Buffer Overflow

  - Attackers can execute arbitrary code by sending a specially crafted packed to the Runetime.exe service of Sielco Sistem Winlog [39];
  - Deploy a Snort rule that detects this specially crafted packet

- DaqFactory HMI NETB Request Overflow
  - An attacker can exploit a stack buffer overflow by sending a crafted 'NETB' request to port 20034 of Azeotech DaqFactory products [40];
  - Deploy a Snort rule that detects this specially crafted packet

Requirements

- Understand how these attacks work
- Deploy IDS rules that verify these attacks

Implementation

For these attacks, three SourceFire certified Snort rules were deployed on the ICS-Security-Appliance's Snort implementation.

- DATAC RealWin SCADA Server Buffer Overflow: `alert tcp $EXTERNAL_NET any -> $HOME_NET 910 (msg:"PROTOCOL-SCADA DATAC RealWin System buffer overflow attempt"; flow:to_server,established; content:"|10 23|Tg"; depth:4; byte_test:4,>,700,0,relative,little; content:"|0A 00 05 00|"; within:4; distance:6; metadata:policy max-detect-ips drop; reference:cve,2011-1563; classtype:attempted-user; sid:24481; rev:7;)`
- Sielco Sistemi Winlog Buffer Overflow: `alert tcp $EXTERNAL_NET any -> $HOME_NET 46823 (msg:"PROTOCOL-SCADA Sielco Sistemi Winlog Pro stack buffer overflow attempt"; flow:to_server,established; content:"|02 01 01|"; depth:3; isdataat:70,relative; metadata:policy max-detect-ips drop; reference:bugtraq,45813; reference:cve,2011-0517; classtype:attempted-admin; sid:21491; rev:7;)`
- DaqFactory HMI NETB Request Overflow: `alert udp $EXTERNAL_NET any -> $HOME_NET 20034 (msg:"PROTOCOL-SCADA DAQFactory NETB protcol stack overflow attempt"; flow:to_server; dsize:>370; content:"NETB"; depth:4; content:"|00|"; within:6; distance:230; metadata:policy max-detect-ips drop; reference:cve,2011-3492;`

```
reference:url,aluigi.altervista.org/adv/daqfactory_1-adv.txt;
classtype:attempted-admin; sid:20176; rev:7;)
```

An assessment of this Use-case can be seen in Section 5.2.11

4.8.17 Internet Accessible Device

Attackers may gain access to industrial environments through systems that are directly exposed to the internet (password authentication may be targeted and compromised – brute forcing maybe)

Use Case

- Detect if a host is exposed to the internet
  - Verify if host exists on baseline;
  - Listen to the traffic going in and out from the host;
  - Verify if the host should be connected to the internet (e.g The host is within a protected netwok);
  - Verify if the host is receiving or sending traffic to outside the protected network.

Requirements

- Establish normal traffic
- Understand what communicating with the internet looks like from a traffic perspective.

Implementation

Implementing this use case, concretely, was not possible. Although, leveraging the *Unusual Endpoints* (Sectioon 4.8.1) use case, should be enough to detect out of the ordinary communications from within the host's protected network (but no assurance can be made with regards to internet communication)

4.9 SIEM

4.9.1 Elastic stack

Filebeat is used for collecting and shipping log files. Filebeat is deployed as an agent on servers that generate logs of interest. It starts an harvester for every log file it is able to detect, from

configured input directories, which then send the data to a specified output. In this particular case, the logs are sipped to Logstash for processing and filtering [16].

Logstash dynamically aggregates the data from different sources, processes this data and, if necessary, filters it, to then output to a desired destination as documents. The filtering capabilities help enrich the data by transforming it according to necessity or separating information into different fields for better visualization [17]. For this implementation, Logstash receives the log data from Filebeat, applies some filtering and transformation mechanisms to better organize and enhance information and outputs the documents to Elasticsearch.

ElasticSearch is the workhorse behind the Elastic Stack. It receives data from the other services, aggregates the data, indexes it as a collection of documents and, once indexed, users can run complex queries on the documents and use aggregations to get relevant information regarding data for a rich post-analysis [15]. In this example, Elasticsearch receives all documents from Logstash, and generates a Logstash index where every document can be found and queried.

Kibana From Kibana, users can create powerful visualizations of their data (in the form of line graph, bar graph, pie charts, heat maps, region maps, coordinate maps, gauge, goals, timeline etc), create dashboards, and manage the Elastic Stack. Kibana provides a visual representation of the data within Elasticsearch which promotes real-time monitoring of intrusions by leveraging IDS logs and other info stored in Elasticsearch documents.

All these components are intertwined and represent the Elastic Stack. Figure 4.5 depicts how all these services relate to each other.

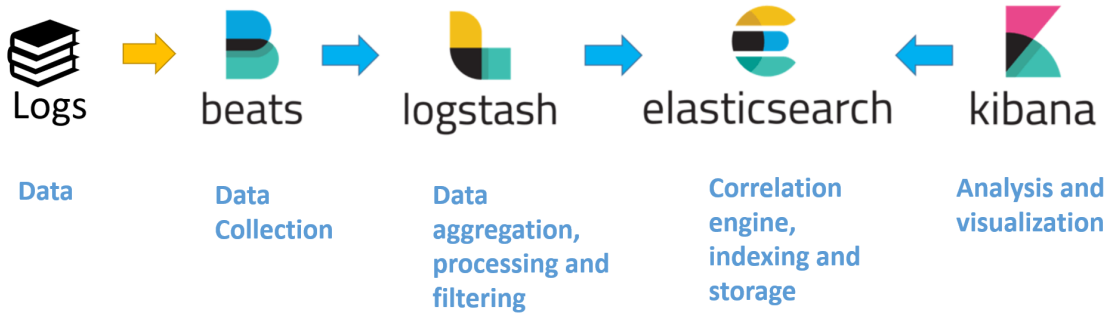

Figure 4.5: Elastic Stack components

4.9.2 Implementation

Logs of interest

Filebeat is configured to take as input several local log directories in order to locate those logs and then send them to an output server being hosted by Logstash. Input directories correspond to the

following logs:

- Snort alerts logs
- Zeek Conn logs
- Zeek Notice logs
- Zeek HTTP logs
- Zeek file logs
- Zeek DNS logs
- Zeek modbus_process logs

Whenever Filebeat locates a log file on a particular directory, it assigns a type tag before outputing the data to Logstash. This allows Logstash to process logs by their type in its pipeline configurations.

In **Logstash** a single configuration file was configured, in order to process and filter every data received from Filebeat. Process and filtering is handled through conditions that, firstly, validate each type tag. As an example, after receiving data from Filebeat, check the log type associated with that data, if it is Zeek-Notice, apply specific filtering for Zeek notice logs.

To illustrate the process behind the ELK stack implemented in the ICS appliance, an example for Zeek's notice log, goes as follows:

1. New Zeek notice log appears in /usr/var/log/zeek/current/notice.log;
2. Filebeat locates log file in /usr/var/log/zeek/current/notice.log;
3. Filebeat assigns "Zeek-Notice" type tag to the located log;
4. Filebeat outputs this log data to Logstash server on port 4045;
5. Logstash listens on port 4045 for Filebeat messages;
6. Logstash receives message on port 4045 with log data;
7. Logstash checks if log type is "Zeek-Notice" and applies the necessary filtering;
8. Logstash outputs the enriched data to the Elasticsearch server;
9. Elasticsearch stores the data has a document on created Logstash index.
10. Kibana acts has a web visual aid of this data

11. Kibana queries all the necessary documents to generate visualization graphs, images and dashboards, as per implementation.

Filtering and transforming data in Logstash allows data coming in from Filebeat in tab separated files, into a more organized Elastic interpretable language - JSON. This is true for every log referred in Section 4.9.2

Dashboard and Visualizations

An ICS-Security-Appliance SIEM dashboard was set up. By resorting to the **Kibana** service within the ELK Stack, it was possible to create some visualizations that, when brought together, illustrate an example Dashboard of security related events on the monitored network. As an example, Figure 4.6, depicts some of these visualizations.

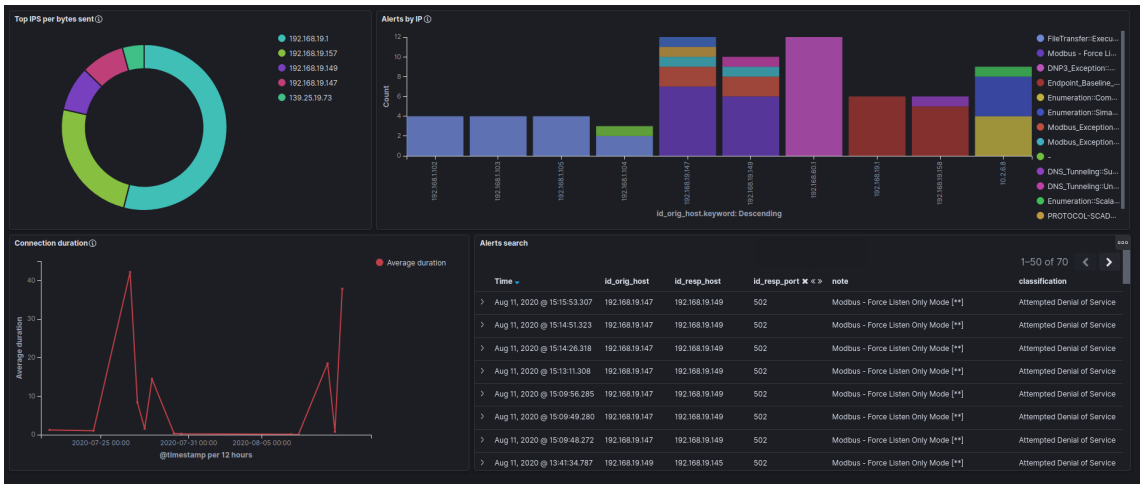

Figure 4.6: Dashboard with some relevant visualizations

The visualization in the top left corner of the dashboard, as can be seen in Figure 4.7, represents a pie chart that showcases the top 5 IPS that send the most amount of bytes per connection - ”the top talkers”. With this information it is possible to visualize which hosts are responsible for generating the most traffic on the monitored network. For example, if there are unrecognized hosts among the network’s top talkers, it could be cause for concern, as it may indicate that a denial of service attack is happening on the network.

On the top right corner we can see a bar graph (Figure 4.8), created to show the amount of alerts being produced by each host on the monitoring network. The colors represent the type of the alert that was generated, each bar represent the IP responsible for triggering that alert, while the size of the bar represent the amount of times that alert was triggered. It’s presence is paramount in order to promote network visibility. A security analyst can quickly obtain intrusion context from the network being monitored and act accordingly.

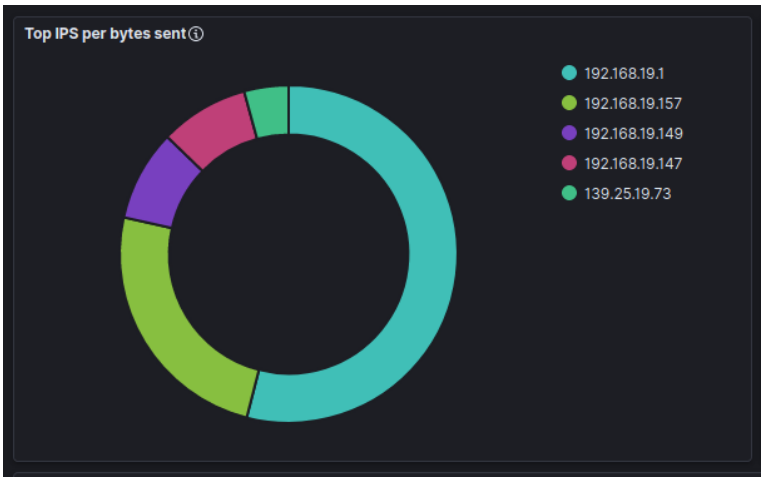

Figure 4.7: Visualization representing a pie chart with the top IPS per bytes sent

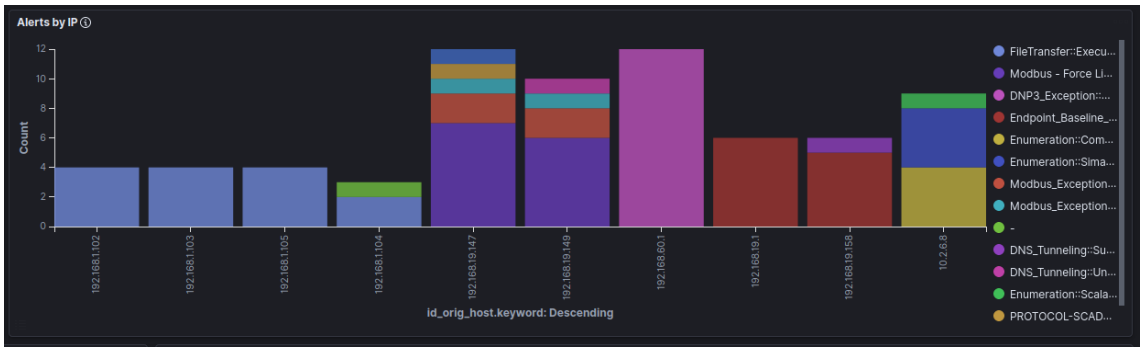

Figure 4.8: Visualization representing a bar graph with a count of attacks from each IP

On the lower left corner, there is a line graph (Figure 4.9) portraying the average duration from all communications. It could be interesting to assess weather or not any new connections exceed the average duration - this could indicate malicious activity, as host connections would be longer than usual.

On the bottom right corner, a table (Figure 4.10 depicting all the attacks that have occurred (denoted by the ”note” field), the hosts involved in the communication, the port on which the connection took place and a ”classification” field for alerts originating from Snort. Once again, providing real-time context to a security analyst.

Another dashboard, more specific to industrial process can be seen in Figure 4.11. This dashboard represents industrial process pertaining to the Modbus protocol and the devices that issue and receive commands through its use. Firstly, it portrays a search table with Modbus industrial process data, specifically, register and coil memory values being read and written. A search table depicting all the Modbus related alerts. A cloud graph representing the top 5 most relevant Modbus Client hosts (who issues Modbus instructions). And, lastly, a data table representing the amount of connection that were set-up between Modbus Client and Modbus Server hosts.

Figure 4.9: Visualization representing a line graph with the average connection duration through time

Alerts search

1–50 of 70

Time	id_orig_host	id_resp_host	id_resp_port	note	classification
Aug 11, 2020 @ 15:15:53.307	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:14:51.323	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:14:26.318	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:13:11.308	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:09:56.285	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:09:49.280	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 15:09:48.272	192.168.19.147	192.168.19.149	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service
Aug 11, 2020 @ 13:41:34.787	192.168.19.149	192.168.19.145	502	Modbus - Force Listen Only Mode [**]	Attempted Denial of Service

Figure 4.10: Visualization representing an ElasticSearch alert search table

Correlation Use Cases

Although it was not possible to implement an automated process for correlating events using methods available within the Elastic Stack, the following use-cases entail potential examples to correlate log information and IDS alerts to fortify or diminish the proof of intrusion.

First Example: Tampering with process values of a Modbus device

- Modbus memory values from a device, **P**, are stored on the SIEM;
- Host **A** (Attacker) executes an unusual control instruction to host **P** that writes to multiple coils;

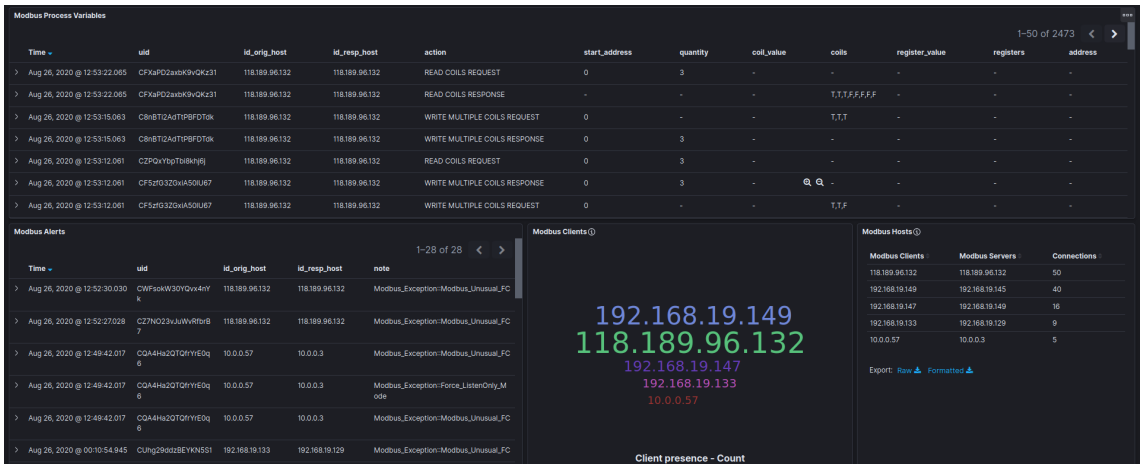

Figure 4.11: Dashboard with some relevant Modbus visualizations

- Alert for unusual instruction appears on SIEM
- A new entry for written coil values **P** appears on the SIEM
- Correlating these events we can assume that host **A** is responsible for altering process values as there was an alert for an unusual control instruction and, all of a sudden, device **P** know registered different coil values on the SIEM.

Second Example: Enumeration of PLC and Engineering Workstation lead to control instruction being sent to the same PLC from the same Engineering Workstation:

- Some host, **A** (Attacker), executes enumeration script on another host, **P** (s7-300 PLC for example);
- Host **A**, executes enumeration on another host, **E** (Engineering Workstation)
- Host **E**, executes *upload program logic* instruction to host **P**
- Correlating all these events, it is probable that the Host **A** was responsible for compromising host **E** to then issue an *upload program logic* instruction to Host **P**. It is very suspicious to have host **E** execute a control instruction, right after being enumerated along with host **P**.

Third Example: Attacking a control device (i.e. PLC), leads to the device returning a downed/unavailable state:

- Device state appears as down on SIEM;
- Within a relatively close time-frame, an *Endpoint deviation* alert from an unknown machine communicating with said device was identified on the SIEM;

- Correlating both these events, it is possible that the unknown machine, responsible for communicating with the device, on out of the ordinary circumstances, has caused the control device to become unavailable.

Chapter 5

Evaluating the ICS-Security-Appliance

The focus of this chapter lies in discussing the potential of the implemented ICS-Security-Appliance. It depicts the results from the detection and correlation use cases considered in the previous chapter, a discussion on the limitations of the baselining, SIEM and detection, and an analysis on the ICS-Security-Appliance strengths while also discussing its short-comings.

5.1 Baselining

When it comes to the baselining mechanisms, some points need to be addressed. It is imperative to have a lot of control over your monitoring network, understand how machines relate to each other to establish a normal network behavior and acknowledge the presence of intrusions or nefarious operation within your environment. To that end, this implementation is relatively straight-forward, in the sense that, it only requires a single execution of Bropy to build a baseline of interest.

Baselining traffic on these environments should only happen once, considering that there shouldn't be too many changes on the network, given the deterministic and constant nature of ICS environments. Alas, if there is no guarantee that traffic is benign before being baselined, the baselined mechanism could consider malicious activity as part of normal behavior, which poses a problem - It wouldn't be possible to detect deviations from attacker activity because it is already considered typical behavior.

It is possible to execute baseline mechanism more than once, if new machines and devices have been added to the network, although it is not advised. Manual entries should be added to the previously generated baseline files instead.

When assessing each baseline file:

- Endpoints

  - Exceedingly useful to detect deviations from normal communication.
  - When a host communicates with a destination host on a destination port through a specific protocol not baselined, an alert is triggered.
  - Not developed by the author of this dissertation, only some changes were introduced for better verbosity

- Route Table
  - Straight-forward file, only containing the IP:MAC pair
  - Particularly useful for confirming, upon parsing ARP messages, if the Mac address for a specific IP remains the same

- Control Hosts
  - Only S7comm ”Stop”, ”Program logic Upload”, ”Program logic Download” commands and IEC 61850 MMS messages are baselined
  - DNP3 and Modbus instructions could also be included in this baseline file, but a separate file has been generated for them.
  - IEC 61850 MMS messages are not well baselined. It was not possible to extract the correct messages from tcp packets as the position on which the message was present, could change depending on encapsulation.

- Devices
  - The manually created device file may be redundant. If we know what equipment (and their corresponding MLFBs) are present on a network, it is very likely that we know which IP addresses are assigned to them.
  - Not all industrial equipment send MLFB information when communicating, only those that send this information can be baselined as devices (All hosts are still baselined as endpoints)

- DNP3
  - Very useful for verifying which IPs issue a specific DNP3 command to a destination IP.
  - Allows for easy detection of unusual instructions being sent to DNP3 devices

- Modbus
  - Much like the DNP3 example, this baseline file is very useful for checking which IPs issue a specific Modbus command to a destination IP.
  - Allows for easy detection of unusual instructions being sent to Modbus devices

- Credentials
  - Tests conducted when detecting the usage of default credentials only verified one entry of the baseline file
  - Even so, it is possible to extrapolate and affirm that, regardless of the credentials or their corresponding devices, that particular use case would still work

In a more favourable setting, the best way to conduct a baseline of traffic would be to have a triage period on which Zeek would limit itself to listening and registering traffic as implemented on a sterile environment, at which point Bropy would execute and generate its baseline files. As it stands, Zeek ensures some degree of detection even before traffic is baselined, which produces a great amount of false positives.

5.2 Use-Case detection Results

For the sake of readability, only some examples will be completely showcased. It is arguably too cumbersome to depict every time a use-case is able to detect the alert as is expected - for that, Table 5.1 describes an overview of which use-cases worked, which use-cases only worked under specific circumstances and which use-cases did not work at all. Along with the corresponding IDS mechanism responsible for triggering the alerts. For the use-cases that require more insight and explanation, the following sub sections also discuss upon those topics.

5.2.1 Unusual endpoint connections

This example represents the alerts generated when a particular connection goes against what is baselined. Two machines never communicate under these specific conditions, and thus an alert is triggered.

ICMP communication between host 192.168.19.155 (ics-appliance) and 192.168.19.1 was never baselined, as can be seen in Figure 5.1, where only one machine, the ics-appliance, has been registered to have been present on the network. Thereforee, in this use case (4.8.1), it was possible to produce an alert by sending ICMP packets to 192.168.19.1 from the ics-appliance. Figure 5.2 depicts the transmission of ICMP packets to the endpoint 192.168.19.1 in order to produce an alert, which can be verified in Figure 5.3

As previously stated, this particular example, although very useful for a simple and straight forward detection, poses some problems:

- The script used to detect was developed by the creator of Bropy rather than the author of

Use Case		Detected			IDS Tool
		Yes	On some cases	No	
Unusual Endpoints		×			Zeek
Communication Channels and Data Exfiltration	DNS - Uncommon Record Type	×			Zeek
	DNS - Suspicious FQDN	×			
	DNS - Data Exfiltration	×			
	ICMP - PowerShell indication	×			
	ICMP - Mismatch echo data	×			
Discovery Techniques	Siemens-SIMATIC-PLC-S7.nse Nmap script	×			Zeek
	s7-enumerate.nse Nmap script	×			Snort
	Communications Processor Nmap script	×			Zeek
	SCALANCE Nmap script		×		Zeek
	Nmap port scan	×			Zeek
New Executable File			×		Zeek
Man in the Middle		×			Zeek
Default Credentials	Default Credential Login		×		Zeek
	Default Credential Spraying		×		
Program Logic Download			×		Zeek
Program Logic Upload			×		Zeek
Service Stop	S7comm - Service Stop		×		Zeek
	DNP3 - Stop Application	×			
System Firmware	Firmware update			×	-
	Activate firmware update mode	×			Snort
Dangerous Instructions	DNP3 - Disable Unsolicited messages	×			Zeek
	Modbus - Force Listen Only mode		×		Zeek or Snort
Unusual Control Instructions	DNP3	×			Zeek
	Modbus	×			
	IEC 61850 MMS		×		
	S7Comm		×		
Segment Smack		×			Snort
Other ICS Attacks	RealWin SCADA	×			Snort
	Winlog	×			
	DAQ Factory	×			

Table 5.1: Detections

```
ics-appliance@ubuntu:/usr/local/zeek/share/zeek/policy/misc/appliance_scripts/ba
seline$ cat baseline_endpoints.data | grep 192.168.19.155
192.168.19.254  67      udp     192.168.19.149/32,192.168.19.1/32,192.168.19.155
/32,192.168.19.145/32,192.168.19.156/32,255.255.255.255/32,192.168.19.150/32
224.0.0.251     5353    udp     192.168.19.155/32,192.168.19.149/32,192.168.0.7/
32
```

Figure 5.1: Occurrences of connections containing host 192.168.19.155 in baselined file

```
ics-appliance@ubuntu:~$ ping 192.168.19.1
PING 192.168.19.1 (192.168.19.1) 56(84) bytes of data.
^C
--- 192.168.19.1 ping statistics ---
19 packets transmitted, 0 received, 100% packet loss, time 18431ms
```

Figure 5.2: ICMP packets sent from host 192.168.19.155 to host 192.168.19.149

this dissertation;

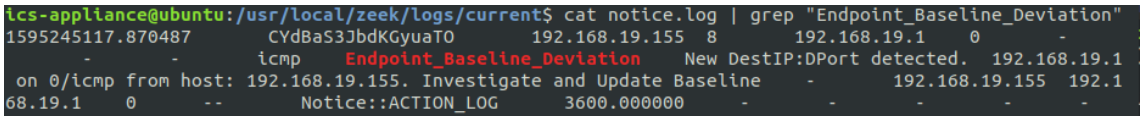

Figure 5.3: Zeek notice logs containing alert for possible deviation from baseline

- It can be a result of false positives in the network. Even in very deterministic environments, it is probable, albeit rare, that machines may establish communication with different endpoints even if these connections did not happen during the baselining phase. Thereforee, it is always important to manually investigate these alerts and, if need be, add another entry for that communication on the baseline file.

5.2.2 Communication Channels and Data Exfiltration

Some limitations in this use case (Section 4.8.2) are important to note.

Verifying the usage of ”TXT” records, although important to detect potential data exfiltration, could be the source of many false positives. ”TXT” records are also used to add descriptive information to requests and also to prevent spoofing by associating with SPF records [8]. Furthermore, to also detect data exfiltration, in this implementation, the method is limited to assessing if the number of times a suspicious fully qualified domain name is accessed by the same IPs more than three times. There is no particular threshold that can be considered fundamentally correct. More importantly, from time to time, there were alerts depicting the occurrence of a suspicious fully qualified domain name, when, in reality, the domain name was large, albeit benign. In these cases, a manual analysis must be taken into consideration or whitelisting these domains at the time of baselining.

For detecting a PowerShell execution for ICMP tunnels, the patterns used to outline the presence of a PowerShell keyword, are not specific enough to guarantee with full certainty that the executed script was specifically used for command and control purposes. The network isn’t always perfect and some packets may be lost, this renders the example of mismatching echo data not overly reliable.

Of course, if it is possible to correlate all these alerts as they simultaneously occur, the amount of certainty that nefarious activity is happening on the network has more weight of truth.

5.2.3 Discovery Techniques

For this example (Section 4.8.3), three NMAP scripts were executed from an attacking machine, represented by **10.2.6.8**, for enumerating three different devices: A *Siemens SCALANCE industrial*

switch represented by **10.2.100.118** and *Siemens SIMATIC-S7 PLC device* with a *communication processor*, both represented by **10.2.100.115**.

As we can see in Figure 5.4, it was possible to detect the requests for "**/Portal/Portal.mwsl**" and "**/docs/cplugError.html/**", as well as in Figure 5.5 the requests for "**/Portal0000.htm**" and **/__Additional**", coming from attacking host **10.2.6.8** running the NMAP scripts.

```
ics-appliance@ubuntu:/usr/local/zeek/logs/current$ cat notice.log | grep Enumeration::Si
matic_S7
1595862222.471105       Cdn3FB45xQf45PhtN1      10.2.6.8        47322   10.2.100.115   8
0       -       -       -       tcp     Enumeration::Simatic_S7 Detected '/Portal/Portal
.mwsl' on http request. Possible SIMATIC PLC enumeration technique from 10.2.6.8 to 10.2
.100.115 on port 80/tcp -       10.2.6.8        10.2.100.115    80      -       -       N
otice::ACTION_LOG       3600.000000     -       -       -       -       -
1595862222.691821       CninPI2x67vV8Wkv        10.2.6.8        47324   10.2.100.115   8
0       -       -       -       tcp     Enumeration::Simatic_S7 Detected '/docs/cplugErr
or.html/' on http request. Possible SIMATIC PLC enumeration technique from 10.2.6.8 to 1
0.2.100.115 on port 80/tcp      -       10.2.6.8        10.2.100.115    80      -       -
Notice::ACTION_LOG      3600.000000     -       -       -       -       -
```

Figure 5.4: Zeek notice logs containing the alert for NMAP script Siemens-SIMATIC-PLC-S7.nse, executed from attacker on 10.2.6.8 to device on 10.2.100.115

```
ics-appliance@ubuntu:/usr/local/zeek/logs/current$ cat notice.log | grep Enumeration::Communicat
ions_Processor
1595855303.095207       Cg0c5F2XAbYZGkVqh3      10.2.6.8        47204   10.2.100.115    80      -
-       -       tcp     Enumeration::Communications_Processor   Detected '/Portal0000.htm' on ht
tp request. Possible Communications Processor enumeration technique from 10.2.6.8 to 10.2.100.11
5 on port 80/tcp        -       10.2.6.8        10.2.100.115    80      -       -       Notice::
ACTION_LOG      3600.000000     -       -       -       -       -
1595855303.124843       C9XA0l2d0Pk2Qmv0gf      10.2.6.8        47206   10.2.100.115    80      -
-       -       tcp     Enumeration::Communications_Processor   Detected '/__Additional' on http
 request. Possible Communications Processor enumeration technique from 10.2.6.8 to 10.2.100.115
on port 80/tcp  -       10.2.6.8        10.2.100.115    80      -       -       Notice::ACTION_L
OG      3600.000000     -       -       -       -       -
```

Figure 5.5: Zeek notice logs containing the alert for NMAP script Siemens-CommunicationsProcessor.nse, executed from attacker on 10.2.6.8 to device on 10.2.100.115

To further solidify the proof of malicious behavior, Zeek's HTTP logs indicate that, indeed, the NMAP software was used. When analysing records with the same **UID** (Session ID) as the alerted records in previously discussed notice logs, it is possible to verify this.

Session id **Cdn3FB45xQf45PhtN1** and **CninPI2x67vV8Wkv** belong to the communication between **10.2.6.8** and **10.2.100.115** when the execution of SIMATIC-S7 NMAP enumeration script took place, as can be verified in Figure 5.6 and Figure 5.7 respectively.

```
ics-appliance@ubuntu:/usr/local/zeek/logs/current$ cat http.log | grep Cdn3FB45xQf45PhtN
1
1595862222.471105       Cdn3FB45xQf45PhtN1      10.2.6.8        47322   10.2.100.115   8
0       1       GET     10.2.100.115    /Portal/Portal.mwsl     -       1.1     Mozilla/
5.0 (compatible; Nmap Scripting Engine; https://nmap.org/book/nse.html) -       0       9
227     200     OK      -       -       (empty) -       -       -       -       -       -
FNRDDjgOxnR7nWNM3       -       text/html
```

Figure 5.6: Zeek's HTTP log for session id Cdn3FB45xQf45PhtN1 where Nmap presence has been registered

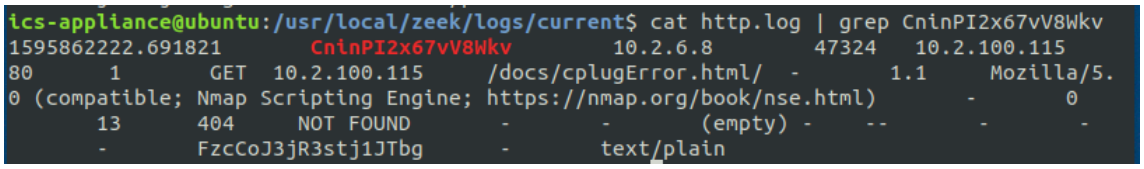

Figure 5.7: Zeek's HTTP log for session id CninPI2x67vV8Wkv where Nmap presence has been registered

With regards to the SCALANCE modules enumeration, it was not possible to obtain real results. The OID returned after executing the NMAP enumeration script is different than what was expected (1.3.6.1.6.3.15.1.1.4.0). Regardless, in order to simulate and show the potentiality of this example, the detection mechanism was altered to detect the presence of the unexpected OID from the SNMP report, as Figure 5.8 demonstrates.

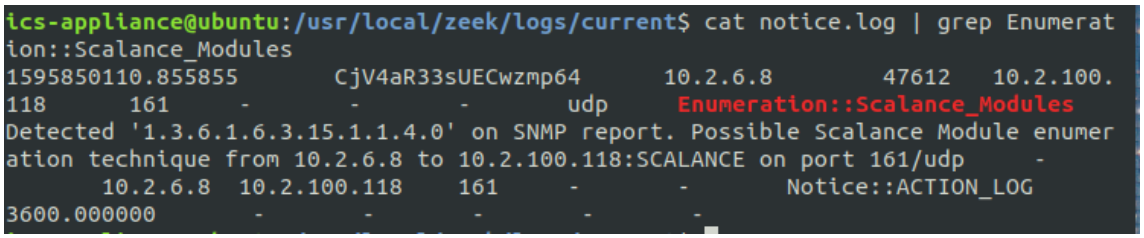

Figure 5.8: Zeek notice logs containing the alert for OID 1.3.6.1.6.3.15.1.1.4.0 in SNMP report

5.2.4 New Executable Files

Originally, for this use case (Section 4.8.4), the plan was to have an attacking machine transfer a file to a victim machine, extract the file's MIME type and assess if it corresponds to an executable file. This was not possible since the tested mechanisms for file transfer encrypt the file. Conversely, a packet capture publicly available for testing Zeek implementations was utilized to simulate a file transfer. Figure 5.9 depicts the alert after executing the packet capture.

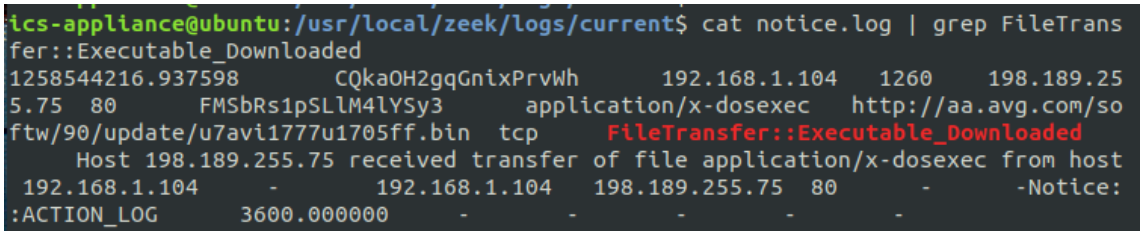

Figure 5.9: Zeek notice log containing the alert for detected MIME type that represents an executable file

As far as was able to determine, MIME types are usually only attached to a file's metadata on HTTP messages. This reduces the applicability of this example significantly, since HTTP is not the only mechanism for transferring files. Zeek, as used within the ICS-Security-Appliance, can,

in fact, extract files from any file transfer protocol (HTTP, FTP, SMTP, among others) so this topic could be discussed and implemented further to allow for better coverage of detecting executable file types circulating the network.

5.2.5 Default Credentials

For both *Default Credential Login* and *Default Credential Spraying*, the implementation solely works with Siemens devices that only support HTTP, stemming from the lack of encryption on HTTP communication (i.e. s7-300 devices). When testing this implementation on recent versions of Siemens s7-1200 and s7-1500 devices, the HTTP content was not parsable, given the fact that these devices upgrade HTTP traffic to its secure counterpart, HTTPS, for login.

The web server is disabled in the default setting of the device's CPU, but it can be enabled in the web server's configurations (as seen in Figure 5.10), also, it is recommended to only allow access with HTTPS. But, as stated before, this approach hinders the detection of default credentials coursing around the network, although with an added layer of security.

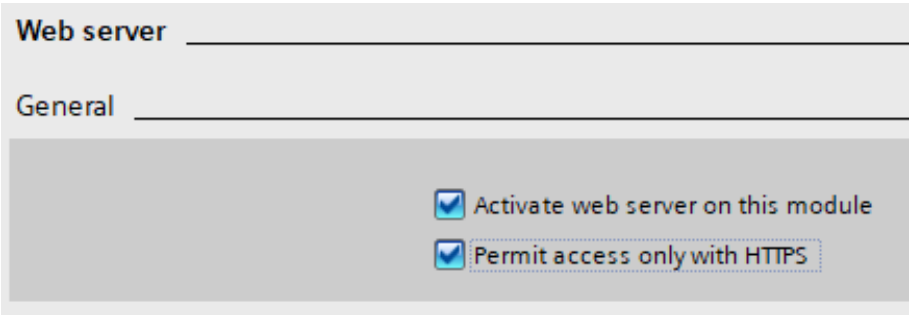

Figure 5.10: Web server configuration for Siemens devices - TIA Portal (proprietary software)

A potential way of circumventing this difficulty, yet retaining the value for using a secure transmission, would be to, possibly, allow the ICS-Security-Appliance to decipher the HTTPS traffic to look at its contents. Figure 5.11 depicts a possible representation of the proposed alternative to detect default credentials on secure transmissions.

Client sends an encrypted HTTPS message to the ICS-Security-Appliance, which deciphers it with key "C", revealing a log-in using "admin" as user and "admin" as password. The ICS-Security-Appliance then ciphers it again to forward it to its designated target, the device (on which the web server is running).

This topic goes beyond the focus of the thesis, but it was important to discuss an alternative to, later, extend this use-case, even if it isn't entirely representative of a real-case implementation. It is just speculation.

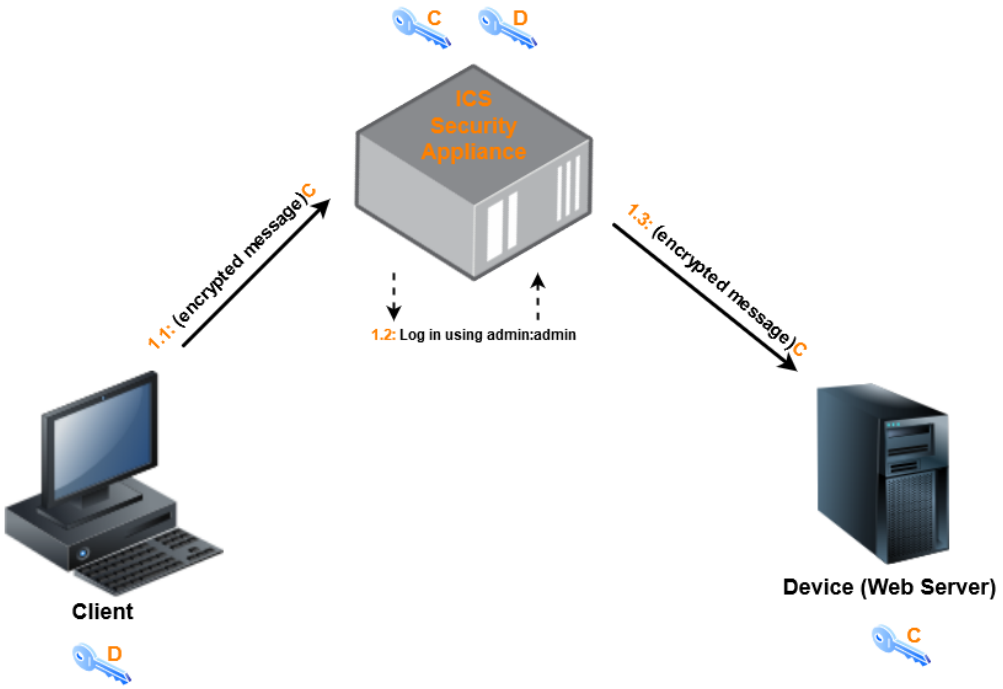

Figure 5.11: Visual representation of potential solution for use case

5.2.6 Program Logic Download, Upload and Service Stop

In this instance (Sections 4.8.8, 4.8.7 and 4.8.9), for both Program Logic Download, Program Logic Upload and Service Stop, the detection is limited to how visible the S7comm protocol is within the monitoring environment. On traffic where S7Comm content is encapsulated with other protocols, it is not possible to find specific strings in the payload that could dictate the occurrence of either of the previous use-cases. S7comm traffic must be in plain text for this implementation to work. Therefore, for the tests that worked, a Siemens s7-300 device was being used.

5.2.7 System Firmware - Firmware Update

The original plan for this use-case was to leverage Zeek's file analysis framework to obtain an hash of the firmware file being transferred on the network. By extracting the hash value of a malicious update file, it would have been possible to compare it to the hash value of a benign update file for the same firmware version, thus providing evidence of intrusion.

This use-case did not work out as expected, Zeek's file extraction framework was not able to detect the firmware update file on the network, let alone, obtain it's hash value.

5.2.8 Force Listen Only mode

Host **192.168.19.147**, the attacking machine, posing as a Modbus Master with the help of a python module, pymodbus, developed by REPL, issued a Modbus *Force Listen Only Mode* instruction to the victim host in **192.168.19.149**. Evidently, Figure 5.12 shows the establishing of a communication between both the attacking host, posing as a Modbus Master, and the victim host, the Modbus Slave and the consequent issuing of a *Force Listen Only Mode* instruction.

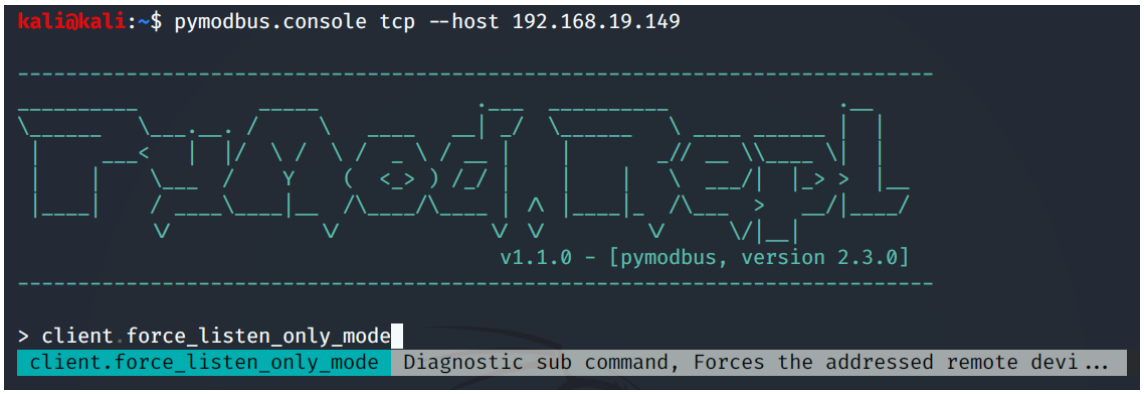

Figure 5.12: Modbus Master in attacking host 192.168.19.147 establishing and issuing Force Listen Only Mode instruction to Modbus Slave victim in 192.168.19.149

After the attacker sent the relevant instruction, the appliance was able to capture the payload and distinguish the *Diagnostics* instruction within it by utilizing Zeek's built in Modbus analyzer. Unfortunately, this analyzer can only distinguish main Modbus function codes 2.3. The alerts are triggered for the presence of a *Diagnostics* function , which could represent a possible *Force Listen Only mode* instruction. Thus, there is never complete certainty with regards to which sub-function is being provided when issuing a *Diagnostics* instruction.

The appliance assumes a *Diagnostics* instruction could be a *Force Listen Only mode*, flags it as a possibility and then alerts, as shown in Figure 5.13. This implementation works, if we assume that a *Diagnostics* instruction is, in fact, a *Force Listen Only mode* instruction, otherwise the amount of false positives could become a fundamental flaw in this use-case.

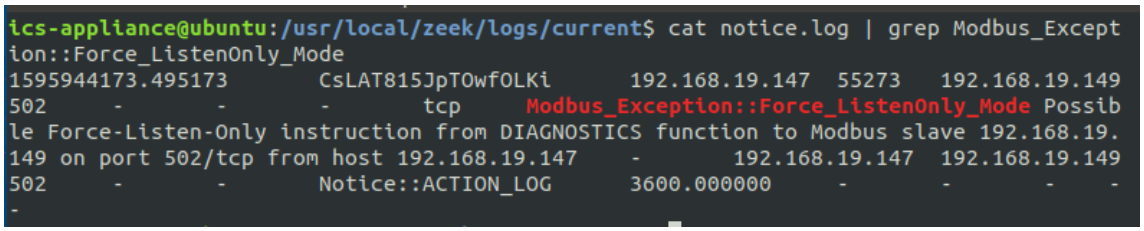

Figure 5.13: Zeek notice logs containing the alert for a possible Force Listen Only instruction inferred from a Diagnostics instruction from Modbus Master 192.169.19.147 to Modbus Slave 192.168.149

To circumvent this limitation, the same attack was detected by the ICS Appliance's Snort

implementation. The veracity in this instance promotes more certainty as Snort is able to verify the sub function *Force Listen Only Mode* within the main function *Diagnostics* of the Modbus payload. Figure 5.14 depicts the alert triggered from the Snort rule deployed in the ICS Appliance.

By correlating both these occurrences as the attack happens, a security analyst can declare, with an explicit degree of certainty, that the attack indeed took place, otherwise, the presence of the *Diagnostics* instruction detected by Zeek, could pose a different outcome other than a *Force Listen Only Mode*. In the future, it could be interesting to assess Modbus sub functions further.

```
ics-appliance@ubuntu:/var/log/snort$ cat snortFull.log | grep "Modbus - Forc
e Listen Only Mode"
08/11-15:15:52.221379  [**] [1:111111111:0] Modbus - Force Listen Only Mode
[**] [Classification: Attempted Denial of Service] [Priority: 1] {TCP} 192.1
68.19.147:49681 -> 192.168.19.149:502
```

Figure 5.14: Snort alert log containing the alert for a possible Force Listen Only instruction from Modbus Master 192.169.19.147 to Modbus Slave 192.168.149

5.2.9 Unusual Control Instructions - S7Comm

In this use case (Section 4.8.15), for the S7Comm protocol, the implementation only verifies the occurrence of either a *program logic download*, *program logic upload* or a *service stop* on the S7Comm payload. Thus, only these instructions are baselined and understandable by the ICS-Security-Appliance. On cases where there are different S7Comm instructions being issued to the control devices, the ICS appliance cannot single them out. Furthermore, much like in Section 5.2.6 - ”Program Logic Download, Upload and Service Stop”, this implementation depends on how visible the S7Comm payload is.

5.2.10 Unusual Control Instructions - IEC 61850 MMS

With regards to the detection of unusual control instruction using the IEC 61850 MMS protocol (Section 4.8.15), the implementation is limited in two aspects:

- It only verifies the 14 main types of MMS PDUs defined by the IEC 61850 standard, which excludes the equally important services sub-types associated with Confirmed and Unconfirmed messages. detailed in Section 2.1.4, that represent the actual actions being requested by the Client to the Server;
- It can only extract the correct MMS PDU type if the MMS data is followed by both TPKT and COTP encapsulation. If more encapsulations are present in the overall TCP packet, then the MMS data is considered to be of an unknown type - when in reality the mechanism extracted another ISO header because it hadn’t been able to reach the MMS PDU data.

Figure 5.15 depicts the only way this implementation is prepared to parse the MMS data. On the TCP packet, only the TPKT and COTP headers are present before the MMS data. This is the formost way of achieving good results. However, Figure 5.16 portrays an example capture where it would not be possible to parse the MMS data - the supposed MMS PDU would be interpreted as *unknown* when in reality it represents the ISO 8327-1 OSI Session Protocol header, which comes right after the COTP header.

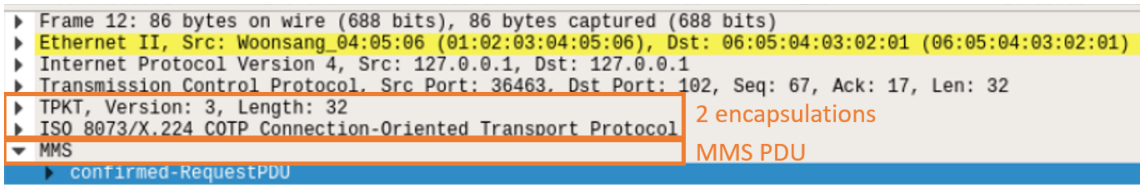

Figure 5.15: confirmed-RequestPDU MMS message taken from wireshark portraying two encapsulations before MMS data

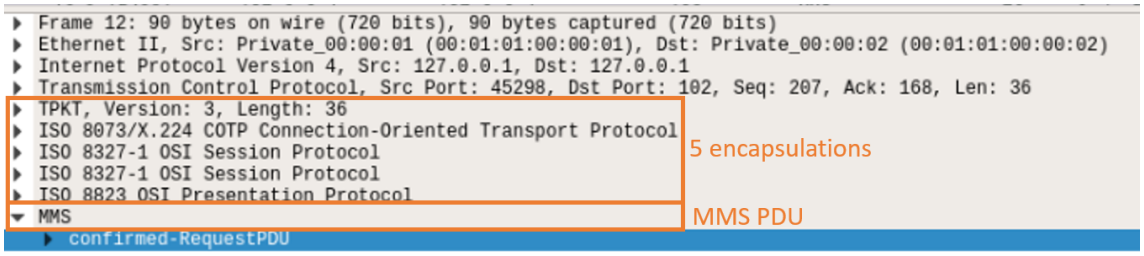

Figure 5.16: confirmed-RequestPDU MMS message taken from wireshark portraying five encapsulations before MMS data

5.2.11 Other ICS attacks

For the attacks introduced in Section 4.8.16 there was no actual work developed by the author of this dissertation with regards to detecting the attacks. The Snort rules utilized for this intention, were taken from available repositories, as was explained in the Base State of the ICS-Security-Appliance in Section 4.2.

These attacks were just a sample of the many ICS related modules the Metasploit attacking tool has access too. To avoid executing every attack and testing every single rule available, just these three examples were taken into consideration in order to enforce the idea behind implementing Snort on an integrated IDS solution for a straightforward mechanism of intrusion detection.

Other attack examples not executable through the Metasploit tool, are also detectable through the rules introduced in Section 4.2. It falls on the person responsible for deploying the ICS-Security-Appliance to know which requirements the target network has, in order to customize and utilize the correct rules for the environment.

5.3 SIEM

Every event of interest is registered in the ELK Stack platform, which allows security analyst to check for potential problems. As mentioned in Section 4.9.2, some Zeek and Snort logs are fed into the SIEM. It is easier to correlate events that, by themselves, wouldn't be considered an actual intrusion.

5.3.1 Dashboards and Visualization

Having relevant dashboards according to necessity is imperative to grant visibility into the monitored environment and quickly deal with issues as they arise.

From the configured dashboards, the average connection duration line graph is, arguably, the least important. The initial idea was to establish a line for the average duration of every connection, and assess if, at any given time, a particular connection duration surpassed the line threshold of average duration. This is obviously not a good approach, as new duration values influence the line representation. What could have been a better approach for this problem, would be to establish the average connection duration at the time of baselining, and set those values as the threshold. The other visualisation graphs and tables are representative of important information across the monitored network.

In a more favourable setting, it would have been great to represent every process variable of every device to ascertain the welfare of industrial process while looking for new, unusual, values appearing on the SIEM that may represent tempering with device's process.

5.3.2 Correlation Use Cases

With regards to the first correlation use-case in Section 4.9.2, Tempering with process values, a possible way to assess its occurrence when analysing the information present in the SIEM would be to verify, firstly, an alerted unusual instruction being sent to a Modbus server, after that, verifying on the Modbus process variables table, if any requests for writing or reading information have taken place.

Figure 5.17 represents a search table with two Modbus related alerts. Two unusual instructions were sent from the host 118.189.96.132 to itself (note: for this test the same host represents the Modbus Client and Server), which means, this Modbus connection is out of the ordinary, as it probably hasn't been baselined. Highlighted in blue and orange, the unique connection ID for each of these occurrences will help identify exactly what happened at process level when searching for these IDs on the Modbus process variables search table. Figure 5.18, depicts the Modbus

process variables search table, containing, as expected, both the connections, highlighted in blue and orange, related to the previously mentioned alerts. For the orange connection, there was a request for writing three coils with the value ”F”, to which a response of ”3” written coils was given. For the blue connection a request for reading ”3” coils was sent, and a response with all the available coils as ”F” was returned.

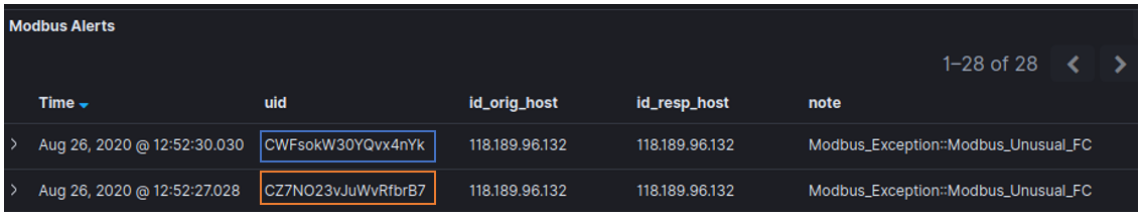

Figure 5.17: Modbus alerts for an unusual function code on the SIEM with connection ids highlighted

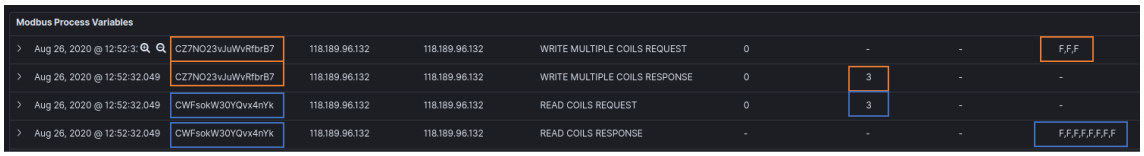

Figure 5.18: Modbus process variable wirte and read requets on the SIEM with connection ids highlighted

This example showcases the potential of having a SIEM solution for aggregating and analysing all log and data information from the monitoring network. It is possible to go as deep as the industrial process itself, and assess deviations from the normal operation. An alert is triggered, and a more thorough assessment is ensured.

Although no automated process for correlation was implemented, within the Elastic Stack available services, the framework for building them has been set up for future implementation.

5.4 ICS-Security-Appliance limitations and strengths

As is it right now, the appliance still needs a lot of manual intervention and monitoring. It is necessary to look at some of the alerts to understand if they really entail an intrusion or if baseline files require readjustment.

The ICS-Security-Appliance is installed on a spanning-port, which means that, it is fundamentally flawed with regards to detection on every level of the purdue model. Even so, it is imperative to place it on a specific vantage point that would promote the most amount of network context so that the appliance could have more visibility. Theoretically, placing the appliance at the industrial control area (comprising both the site manufacturing operation/control and supervisory control layer) and at the Enterprise network (comprising both the enterprise network and Site business

planning/logistics layer) would be the best approach. To avoid having to deploy two distinct appliances, the implementation could have benefited from a centralized solution that would receive a copy of traffic seen on sensors deployed on spanning ports of both those areas of interest. The sensors would receive a copy of traffic and send it to the ICS-Security-Appliance for detection and monitoring at a remote host.

With the ICS-Security-Appliance, security analysts and responders, responsible for ensuring a secure perimeter around industrial environments, can deploy a very customizable solution specific to their network, that depends on its underlying necessities and process logic. It is also possible to deal with intrusions out of the box, for more common threats within the ICS scope, as represented by the MITRE ATT&CK for ICS [45]. As long as the industrial environments, on which the appliance would be deployed, are running devices that influence and are influenced by some of the use-cases referred in chapter 4, we can be sure that detection is possible and works for the examples evaluated in the previous sections.

The ability to have all the most important logs being displayed and accessible on a SIEM solution is a very powerful approach to guarantee visibility on the network being monitored. The use-cases for SIEM correlation, previously introduced, help pave the way for further correlation and potential development on top of the appliance, as it is very customizable.

Aside from ensuring detection for certain attack examples, specifically ICS related but also other common pre-ICS-compromise related, the fact that it establishes a straight-forward framework for further development and support, effectively fine tuning it to specific industrial process needs, is very beneficial and valuable from a long-term security perspective.

New IDS rules constantly created by the community can be easily deployed, and other detection policies can be implemented on top of existing ones, while leveraging all the available scripts and logging mechanisms provided by Zeek, to help focus the detection depending on the exact type of industrial process happening on the network - Although, as of right now, it would be significantly easier to do so on DNP3 and Modbus communications.

Chapter 6

Conclusions and Future Work

In this final chapter, the conclusions drawn with the realization of this project will be presented, starting with a short summary of the work carried out and ending with a section where future work possibilities will be presented, namely, what can still be improved or developed in the future.

6.1 Summary

The lack of security mechanisms on industrial control systems poses a real-world problem that could give rise to potentially disastrous consequences, provided systems have been breached. It is of great importance to secure these systems as best as possible, by applying many security controls, and doing so effectively. Thus, on it's completion, this dissertation has provided an assessment of potential security mechanisms to be applied on these environments (Section 3.2 of Chapter 3), as well as a developed security solution for attack detection and monitoring that leverages both Zeek-IDS and Snort-IDS for detection, Bropy for baselining traffic and the ELK stack for monitoring (Chapter 4).

As the dissertation progresses, there is a discussion regarding the thought process, conducted tests and research that helped formulate a final decision concerning the implementation of an integrated intrusion detection and monitoring solution that helps protect industrial environments from threat actors.

- A possible attack route taken by threat actors to compromise industrial systems is introduced;
- A survey of security mechanisms is presented;
- A final section exhibiting the decision and the proposed solution for an intrusion detection and monitoring solution – ICS-Security-Appliance.

The focal point of the dissertation lies in the explanation and results concerning the implemented ICS-Security-Appliance.

- Sections detailing the ICS-Security-Appliance's architecture and system model to understand the system's functionalities, and a description of the attacker model to understand what the capabilities of a threat actor are.
- The later sections are more in line with implementation, exhibiting every characteristic behind traffic baselining, the intrusion detection use cases and a SIEM configuration, all within the proposed solution.

For detection, the use-cases considered were taken from MITRE's established knowledge base for industrial control systems' attack techniques and procedures, as well as some studied vulnerabilities of industrial protocols. To detect these attacks, three tools were used and extended – Snort-IDS, Zeek-IDS and Bropy.

- Zeek for logging – Aside from the default logging policies (Zeek scripts), other policies for logging relevant information to aid in detection were implemented;
- Bropy for baselining – From previously created Zeek logs, aside from the default baselining python scripts, other python scripts were developed to generate baseline files from traffic;
- Zeek for detection – Policies (Zeek scripts) were implemented to enforce network-level and process-level analysis of traffic to detect potential intrusions;
- Snort for detection – Snort rules were deployed to enforce network-level analysis to detect intrusions from known signatures:

A SIEM solution (ELK Stack) was deployed and configured to ensure network context and monitoring.

- ELK Stack as a SIEM – Most of the important logs and alerts generated from Zeek and Snort were configured to be pushed into the SIEM for posterior analysis. From this data, visualizations and dashboards were created to promote visibility into the environment:

Overall, a great amount of concepts had to be studied and learned to effectively produce the desired implementation, such as understanding industrial protocol concepts; understand and conduct packet inspection; employ regex mechanisms to validate protocol instructions and bytes of interest, among others. Even though the developed security solution ensures detection of certain attack examples (specially ICS-borne) it establishes a streamlined framework for further development,

customization and support. It is highly extendable, as new Snort rules can be easily deployable as they become available (there is a lot of community support for Snort rules, even for ICS related attacks), new Zeek detection policies can be implemented and deployed on it's policy engine to guarantee a better detection surface depending on the exact type of industrial process and network behavior. Thus, effectively fine tuning the solution to respond to the specific production needs.

6.2 Future Work

For future work, it would be interesting to implement more mechanisms for parsing other instructions for the S7Comm protocol (aside from the three instructions mentioned in this dissertation) and the IEC-61850-MMS Standard. Furthermore, ensuring other ICS protocols can be parsed, would also contribute to the applicability and scientific value of this project.

Moreover, the implementation of automatic event correlation at SIEM level, utilizing the available mechanisms within the ELK Stack, would benefit the detection and monitoring results of the developed solution, while finding new and improving existing use-cases.

As far as detection goes, the framework for developing further detection use-cases has been set-up. Thus, it would be of particular interest to continue developing and extending this project to accommodate new detection use-cases for different TTPs within ICS.

Bibliography

[1] Johan Angséus and Rikard Ekbom. Network-based intrusion detection systems for industrial control systems. *Göteborg, Sweden: Institutionen för data-och informationsteknik, Chalmers tekniska högskola*, 2017.

[2] Longe Olumide Babatope, Lawal Babatunde, and Ibitola Ayobami. Strategic sensor placement for intrusion detection in network-based ids. *International Journal of Intelligent Systems and Applications*, 6(2):61, 2014.

[3] Rafael Ramos Regis Barbosa, Ramin Sadre, and Aiko Pras. Flow whitelisting in scada networks. *International journal of critical infrastructure protection*, 6(3-4):150–158, 2013.

[4] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. The operational role of security information and event management systems. *IEEE security & Privacy*, 12(5):35–41, 2014.

[5] John Bigham, David Gamez, and Ning Lu. Safeguarding scada systems with anomaly detection. In *International Workshop on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 171–182. Springer, 2003.

[6] Eric Byres. The air gap: Scada's enduring security myth. *Communications of the ACM*, 56:29–31, 08 2013.

[7] Jeffrey Carr. Snort: Open source network intrusion prevention. `https://www.esecurityplanet.com/network-security/Snort-Open-Source-Network-Intrusion-Prevention-3681296.htm`. Accessed on 25.11.2019.

[8] SANS Internet Storm Center. Sans isc: Dns query length... because size does matter. `https://isc.sans.edu/forums/diary/DNSQueryLengthBecauseSizeDoesMatter/22326/`. Accessed on 16.08.2020.

[9] Inc Cloudfare. What is a denial-of-service (dos) attack? `https://www.cloudflare.com/learning/ddos/glossary/denial-of-service/`. Accessed on 25.08.2020.

[10] M Herrero Collantes and A Lpez Padilla. Protocols and network security in ics infrustructures. *Tech. Rep.*, 2015.

[11] Ihab Darwish, Obinna Igbe, Orhan Celebi, Tarek Saadawi, and Joseph Soryal. Smart grid dnp3 vulnerability analysis and experimentation. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 141–147. IEEE, 2015.

[12] Hervé Debar. An introduction to intrusion-detection systems. *An Introduction to Intrusion-Detection Systems*, 01 2009.

[13] Chris Duffey. Monitoring ics with splunk: Scada, historians, and alarms, oh my! `https://www.splunk.com/en_us/blog/iot/monitoring-ics-with-splunk-scada-historians-and-alarms-oh-my.html`. Accessed on 27.11.2019.

[14] Elastic. Elastic stack. `https://www.elastic.co/products/elastic-stack`. Accessed on 27.11.2019.

[15] Elastic. Elasticsearch. `https://www.elastic.co/what-is/elasticsearch`. Accessed on 27.11.2019.

[16] Elastic. Filebeat overviewedit. `https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-overview.html`. Accessed on 25.07.2020.

[17] Elastic. Logstash: Collect, parse, transform logs. `https://www.elastic.co/logstash`. Accessed on 25.07.2020.

[18] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 5(6):29, 2011.

[19] Greg Farnham and A Atlasis. Detecting dns tunneling. *SANS Institute InfoSec Reading Room*, 9:1–32, 2013.

[20] Mislav Findrik, Paul Smith, Kevin Quill, and Kieran McLaughlin. Plcblockmon: Data logging and extraction on plcs for cyber intrusion detection. *Proceedings of Proceedings of ICS & SCADA*, page 102, 2018.

[21] FortiGuard. Modbus.tcp.force.listen.only. `https://fortiguard.com/encyclopedia/ips/11521/modbus-tcp-force-listen-only`. Accessed on 31.08.2020.

[22] Graylog. What's inside: Graylog features. `https://www.graylog.org/features`. Accessed on 27.11.2019.

[23] Jeffrey J. Guy. dns part ii: visualization. `http://armatum.com/blog/2009/dns-part-ii/`. Accessed on 14.08.2020.

[24] Dina Hadvziosmanovic, Robin Sommer, Emmanuele Zambon, and Pieter H Hartel. Through the eye of the plc: semantic security monitoring for industrial processes. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 126–135. ACM, 2014.

[25] Chris Hankin and Tom Chothia. Availability of open source tool-sets for cni-ics. Technical report, Imperial College London, 2018.

[26] Sinclair D Hansen. *An intrusion detection system for supervisory control and data acquisition systems*. PhD thesis, Queensland University of Technology, 2008.

[27] Derek R Harp and Bengt Gregory-Brown. It/ot convergence-bridging the divide. *NEX DEFENSE*, 2014.

[28] Matt Domko Hashtagcyber. bropy. `https://github.com/hashtagcyber/bropy`, Oct 2017. Accessed on 27.11.2019.

[29] M Horkan. Challenges for ids/ips deployment in industrial control systems. *SANS Institute reading room*, 2015.

[30] Yan Hu, An Yang, Hong Li, Yuyan Sun, and Limin Sun. A survey of intrusion detection on industrial control systems. *International Journal of Distributed Sensor Networks*, 14(8):1550147718794615, 2018.

[31] Carl M Hurd and Michael V McCarty. A survey of security tools for the industrial control system environment. Technical report, Idaho National Lab.(INL), Idaho Falls, ID (United States), 2017.

[32] William Jardine, Sylvain Frey, Benjamin Green, and Awais Rashid. Senami: Selective non-invasive active monitoring for ics intrusion detection. In *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pages 23–34. ACM, 2016.

[33] Dylan Jenkins. How to hack an iec 61850 system (or protect one). 09 2017.

[34] Ryan Jordan. Snort 2.9.2: Scada preprocessors. `https://blog.snort.org/2012/01/snort-292-scada-preprocessors.html`. Accessed on 25.11.2019.

[35] Barbara Kay. Rules vs. behavioral heuristics vs. data mining vs. machine learning. actually, you want it all., Mar 2018.

[36] Gulshan Kumar. Evaluation metrics for intrusion detection systems-a study. *Evaluation*, 2(11):11–7, 2014.

[37] Robert M Lee, Michael J Assante, and Tim Conway. Tlp: White analysis of the cyber attack on the ukrainian power grid. *E-ISAC, Tech. Rep*, 2016.

[38] Alan Lewis. *The Cambridge handbook of psychology and economic behaviour*. Cambridge University Press, 2008.

[39] MC mc@metasploit.com Luigi Auriemma. Sielco sistemi winlog buffer overflow. https://www.rapid7.com/db/modules/exploit/windows/scada/winlog_runtime. Accessed on 25.08.2020.

[40] mr_me steventhomasseeley@gmail.com Luigi Auriemma. Daqfactory hmi netb request overflow. https://www.rapid7.com/db/modules/exploit/windows/scada/daq_factory_bof. Accessed on 25.08.2020.

[41] Petr Matoušek. Description of iec 61850 communication. Technical report, Faculty of Information Technology BUT, 2018.

[42] MC mc@metasploit.com. Datac realwin scada server buffer overflow. https://www.rapid7.com/db/modules/exploit/windows/scada/realwin. Accessed on 25.08.2020.

[43] Gyorgy Miru. The siemens s7 communication - part 1 general structure. http://gmiru.com/article/s7comm/, Jan 2016. Accessed on 25.08.2020.

[44] Gyorgy Miru. The siemens s7 communication - part 2 job requests and ack data. http://gmiru.com/article/s7comm-part2/, Jun 2017. Accessed on 25.08.2020.

[45] MITRE. Att&ck® for industrial control systems. https://collaborate.mitre.org/attackics/index.php/Main_Page. Accessed on 20.4.2020.

[46] Netgate. pfsense. https://www.pfsense.org/getting-started/. Accessed on 07.12.2019.

[47] Andrew Nicholson, Helge Janicke, and Antonio Cau. Position paper: Safety and security monitoring in ics/scada systems. In *ICS-CSR*, 2014.

[48] Jeyasingam Nivethan and Mauricio Papa. A scada intrusion detection framework that incorporates process semantics. In *Proceedings of the 11th Annual Cyber and Information Security Research Conference*, pages 1–5, 2016.

[49] Luciana Obregon. Secure architecture for industrial control systems. *SANS Institute InfoSec Reading Room*, 2015.

[50] Otw. Scada hacking: Scada protocols (dnp3). https://www.hackers-arise.com/post/2017/02/10/scada-hacking-scada-prortocols-dnp3, Feb 2017. Accessed on 15.10.2019.

[51] Otw. Scada hacking: Scada/ics communication protocols (modbus). https://www.hackers-arise.com/post/2017/01/05/scada-hacking-scadaics-communication-protocols-modbus, Jan 2017. Accessed on 15.10.2019.

[52] Nicholas Pappas. Network ids & ips deployment strategies. *2nd April*, 2008.

[53] Tadeusz Pietraszek. Dnscat. http://tadek.pietraszek.org/projects/DNScat/. Accessed on 14.08.2020.

[54] Proofpoint. Phishing - what it is, emails & attacks. https://www.proofpoint.com/us/threat-reference/phishing, Aug 2020. Accessed on 25.08.2020.

[55] Automated Results Computer Consulting LLC Rich Winslow. What is a data historian? http://www.automatedresults.com/PI/data-historian-overview.aspx. Accessed on 25.08.2020.

[56] Joaquín Rodríquez. Most common attack vector over critical infrastructures. https://www.cipsec.eu/content/most-common-attack-vector-over-critical-infrastructures. Accessed on 27.07.2020.

[57] Juan Enrique Rubio, Cristina Alcaraz, Rodrigo Roman, and Javier Lopez. Current cyber-defense trends in industrial control systems. *Computers & Security*, 87:101561, 2019.

[58] SANS. Anatomy of an ics network attack - security awareness video. https://www.youtube.com/watch?v=_eNB1gq5gbA&t=385s, 2016. Accessed on 31.08.2020.

[59] Xavier Mertens SANS Internet Storm Center. Sans isc: Dns query length... because size does matter. https://isc.sans.edu/forums/diary/DNSQueryLengthBecauseSizeDoesMatter/22326/, Apr 2017. Accessed on 10.04.2020.

[60] Sisco. https://www.sisconet.com/wp-content/uploads/2016/03/mms_abstract_syntax.txt. Accessed on 25.08.2020.

[61] Joseph Slowik. Crashoverride: Reassessing the 2016 ukraine electric power event as a protection-focused attack. *Dragos*, 2019.

[62] Joseph Slowik. Evolution of ics attacks and the prospects for future disruptive events. *Dragos*, 2019.

[63] Stackify. What is telemetry? how telemetry works, benefits, and tutorial. https://stackify.com/telemetry-tutorial/, May 2019. Accessed on 25.11.2019.

[64] Keith Stouffer. Guide to industrial control systems (ics) security. *NIST special publication*, 800(82):16–16, 2011.

[65] Suricata. Suricata. https://suricata-ids.org/. Accessed on 25.11.2019.

[66] TC Synchronization. Channel coding—summary of concept and rationale. *Report Concerning Space Data System Standard. Informational Report CCSDS*, 2012.

[67] Jan Sørensen and Martin Jaatun. An analysis of the manufacturing messaging specification protocol. pages 602–615, 01 2008.

[68] Zeek. base/utils/exec.zeek. `https://docs.zeek.org/en/current/scripts/base/utils/exec.zeek.html`. Accessed on 30.06.2020.

[69] Zeek. The zeek network security monitor. `https://www.zeek.org/`. Accessed on 25.11.2019.