Intrusion Detection for Industrial Control Systems

Gorby Nicolas Kabasele Ndonda

Thesis submitted in partial fulfillment of the requirements for the Degree of Doctor in Applied Sciences

March 2022

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Ramin Sadre **Supervisor**	UCLouvain/ICTEAM, Belgium
Pr. Charles Pecheur **President**	UCLouvain/ICTEAM, Belgium
Pr. Etienne Riviere **Secretary**	UCLouvain/ICTEAM, Belgium
Pr. Radu State	University of Luxembourg, Luxembourg
Pr. Anne Remke	University of Münster, Germany

Intrusion Detection for Industrial Control Systems

by Gorby Nicolas Kabasele Ndonda

ICTEAM
Université catholique de Louvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work was partially supported by the ARC-SDN project.

The greatest danger in times of turbulence is not the turbulences, it's acting with yesterday's logic

PETER DRUCKER

Abstract

Industrial Control Systems (ICS) are computer systems used for monitoring and controlling industrial facilities such as water treatment facilities, power plants, manufacturing factories.... Historically those systems were isolated but for business and management purposes, they got connected to other networks. This connection brought the vulnerabilities of other networks to ICS which resulted in an increasing number of attacks against those systems. Consequently, the security of ICS has become an active research field. To improve the resilience of ICS against cyber-attacks, researchers are focusing on designing protection mechanisms such as Intrusion Detection Systems (IDS). In this thesis, we present two IDS solutions for ICS to consider both the cyber aspect and the physical aspect of ICS. The first solution focuses on the detection of attacks targeting the network infrastructure of the ICS and leverages Software-defined Networking (SDN). SDN is a network paradigm that provides a high-level of flexibility in term of network management. We use this property to improve a technique called Flow-Whitelisting.

The second solution focuses on process-oriented attacks, meaning attacks that target physical processes. The IDS learns the temporal properties of a physical process to detect the disruptions caused by an attack. It analyzes the variables defining the physical process and monitors their behavior over time. Any deviation from the learned temporal properties triggers an alert. Each IDS is evaluated with several scenarios and gives interesting results.

In addition, we provide a network dataset from a real system but also a tool to generate new datasets that can be used for the evaluation of network IDS. To show the usefulness of the dataset, we evaluate several IDS in the literature.

Résumé

Les Systèmes de Contrôle Industriels (ICS) sont des systèmes informatiques utilisés pour surveiller et contrôler l'ensemble des activités se déroulant dans des environments industriels tel que les usines de traitement d'eau, les centrales nucléaires,les usines de fabrication .... Historiquement, ces systèmes étaient isolés mais pour des raisons de business et de management, ils ont été connectés à d'autres réseaux. Cette connection a importé les vulnérabilités de ces autres réseaux dans les ICS en plus de les rendre plus accessibles ce qui a eu pour conséquence une augmentation du nombre d'attaques visant ces systèmes. Par conséquent, la sécurité des ICS est devenu un domaine de recherche de plus en plus actif. Pour améliorer la résilience des ICS contre les cyber-attaques, les chercheurs se sont attelés à mettre au point des mécanismes de protections tel que les Systèmes de Détection d'Intrusions (IDS). Dans cette thèse, nous présentons deux IDS conçus spécifiquement pour les ICS de façon à prendre en compte la partie informatique et la partie physique des ICS. Le premier IDS se focalise sur la détection des attaques qui ciblent le réseau et son infrastructure. Elle tire partie des réseaux définis par logiciel (SDN), un nouveau paradigme de réseau qui rend la gestion du réseau plus fléxible. Nous utilisons cette propriété pour améliorer une technique appelée le Flow-Whitelisting.

La seconde solution sert à detecter les attaques visant les processus industriels controlés par l'ICS. L'IDS apprend les propriétés temporelles de ces processus et détecte les perturbations causées par une attaque. L'IDS analyse les variables définissant le processus et surveille leurs comportements au cours du temps. Une dévation par rapport à ce que l' IDS a appris sur les propriétés temporelles des variables déclenche une alerte. Chaque IDS est évalué à travers plusieurs scénarios et donne des résultats intéressants.

Nous mettons également à disposition, un jeu de donnée réseau provenant d'un vrai système et un outil permettant de générer d'autres jeux de données qui peuvent être utilisés pour l'évaluation d'IDS. Pour montrer l'intérêt de ce jeu de donnés, nous l'évaluons avec plusieurs IDS venant de la littérature.

Acknowledgments

This manuscript is the result of hours of work, readings, discussions but it could not have happen without the involvment of several people for whom I want to express my gratitude. First and foremost, my promoter Ramin Sadre who accepted to go on this head-scratching journey that is a PhD thesis. I would like to thank him for his guidance, his patience, his understanding and the amount of effort he put in the works we collaborated on. I would also like to thank the jury members for reading this thesis and helping me improve it with their comments and questions.

Then I want to thank all my colleagues at the INGI Department. Amazing people who are studious, diligent in their work eager to learn new things but also open-minded and fun to hang out with. I have learned a lot from them and I hope they have learned a bit from me. I feel lucky to have worked in an environment where research activities are obviously important but socializing via the several social events organized during the year, the boardgame nights, the bingewatching sessions,... is also an important aspect of the life of the department. Among the colleagues/friends, I would like to address a special thanks to the other member of the now infamous Club D&D: Fabien, Mathieu, Charles, Marie, Hélène for all the awesome roleplaying sessions and fun activities that helped me a lot to keep my sanity, especially during the pandemic. I would also like to thank all of my friends and my numerous uncles and aunties for their overall support and their presence. It means a lot to me.

Last but not least, I would like to thank my family, my two brothers Gaëtan and Gregory for their joyfulness which is always a blessing to see. My mom, for her unlimited kindness and support throughout these years and my Dad for his advices and who convinced me to start this thesis.

Contents

List of Figures

List of Tables

Acronyms

ARP Address Resolution Protocol

BMS Building Management Systems

DCE Distributed Computing Environment

DCS Distributed Control Systems

DMZ Demilitarized Zone

DNP3 Distributed Network Protocol

DNS Domain Name Server

DPI Deep Packet Inspection

DR Detection Rate

EWMA Exponential Weighted Moving Average

FPR False Positive Rate

HIDS Host-based Intrusion Detection Systems

HMI Human Machine Interface

HTTP Hypertext Tranfer Protocol

HVAC Heating Ventilation and Air Conditioning

ICMP Internet Control Message Protocol

ICS Industrial Control Systems

IDS Intrusion Detection Systems

IP Internet Protocol

IPS Intrusion Prevention Systems

IT Information Technology

ICT Information Control and Technology

IPC Inter-Process Communication

IPT Inter-Packet Time

KDE Kernel Density Estimation

LAN Local Area Network

LLDP Link Layer Discovery Protocol

LMS Least Mean Square

LOF Local Outlier Factor

MAC Media Access Control

MTU Master Terminal Unit

MCP Minimum Cost flow Problem

NFV Network-Function Virtualization

NIDS Network-based Intrusion Detection Systems

NTP Network Time Protocol

NIST National Institute of Standards and Technology

OT Operation Technology

OUI Organizationally Unique Identifier

P4 Programming Protocol-Independent Packet Processor

PISA Protocol-Independent Switch Architecture

PLC Programming Logic Controllers

PS Packet Size

ROC Receiver Operating Characteristic

RPC Remote Procedure Call

RTT Round-Trip Time

RTU Remote Terminal Unit

SCADA Supervisory Control and Data Acquisition

SDN Software-Defined Networking

SNMP Service Network Management Protocol

TCP Transmission Control Protocol

UDP User Datagram Protocol

WAN Wide Area Network

Introduction 1

Among computer systems, Industrial Control Systems (ICS) take a special role because they are used to manage industrial facilities and therefore are directly attached to devices that control physical processes. They are also key components of the so-called critical infrastructures where they control water treatment facilities, power plants, or entire power grids. Over the past years, these systems have been the target of an alarming number of cyber-attacks and despite their importance, ICS are often less secure than ordinary Information and Communication Technology systems (ICT). This is a consequence of developments in recent years that have drastically changed the design and structure of ICS. Originally, ICS were isolated systems relying on proprietary hardware and software. However, for business reasons, the usage of "off-the-shelf" components, such as mainstream software and common IP-based network protocols, has become increasingly popular. At the same time, ICS are nowadays often directly or indirectly connected to the Internet in order to simplify their remote management and the real-time extraction of performance metrics. This development has brought known vulnerabilities of ICT networks to ICS. Consequently, the security of ICS has become an active research field. To improve the resilience of ICS against cyber-attacks, researchers are focusing on designing protection mechanisms such as Intrusion Detection Systems (IDS).

ICS include both a cyber aspect, as they operate using common ICT technology, and a physical aspect, as they interact with the physical world through sensors and actuators. Because of those two aspects, the attack surface is diverse as well as the means that attackers have to disrupt the system. They can either target the physical part or the cyber part. Researchers have addressed the duality of ICS by proposing different detection approaches for each part, and this work also follows this path.

For the cyber part, many existing publications focus on the detection of network attacks but their solutions lack generality as they can only be applied in certain contexts or they have strong requirements. Many of the solutions proposed rely on machine learning techniques and require the collection of data. They focus on specific protocols and/or are too intrusive as they need an active interaction with the system to collect data. In this thesis, we propose, among others, the usage of Software-Defined Networking to achieve efficient

intrusion detection on network level.

Works focusing on the detection of cyber-attacks targeting the physical part are less numerous because it requires that the detection mechanism understands the behavior of the physical processes controlled by the ICS. This thesis presents an IDS that avoids some of the limitations of existing solutions, which are sensitive to noisy data and fail to detect certain types of attacks. It explores a new way to detect process-oriented attacks while achieving interesting results.

In more detail, the contributions of the thesis are:

- *Generation of synthetic ICS network traffic for the evaluation of Flow-based IDS* [NS20]
  Flow-based IDS analyze the characteristics of the network flows occurring in a network in order to detect attacks. Unlike IDS based on deep packet inspection, Flow-based IDS do not rely on an analysis of the *content* of the network communication. This property is particularly interesting for ICS networks, since the latter are known to often use closed and proprietary application protocols. Because of the critical nature of ICS, it is challenging for researchers to get access to datasets that they can use to evaluate their solutions. In this thesis, we create a tool to generate synthetic network flow traces that contain both legitimate and malicious traffic flows. The traces are seeded with empirical traces to make them as realistic as possible. We use traces generated by our tool to evaluate the performance of several Flow-based IDS in ICS environments. The code for this work is available at [Kabd].

- *Using Software-Defined Networking (SDN) for ICS security* [NS18; NS17]
  Flow-Whitelisting is a popular approach in ICS security. It leverages the static nature of ICS where the creation/deletion of network flows are rare events. However, the whitelist has to be carefully prepared: If the list is incomplete, legitimate traffic might be blocked. We propose a Two-level IDS that extends the concept of whitelisting by a dynamic component using SDN. The first level of the IDS is implemented on a P4-programmable switch and acts as a whitelist. The second level is a security engine running on a dedicated host and decides whether a new flow that is not on the whitelist should be accepted. The code for this work is available at [Kabe].

  We also use SDN to implement a low-delay countermeasure to eavesdropping attacks in ICS networks. The SDN-capable network uses a multipath routing strategy that alternates between several paths used by any pair of hosts so that the communication can not be eavesdroppend by an attacker. The network computes disjoint paths to minimize

the eavesdropping risk while ensuring the shorteness of the path to avoid high delay. The code for this work is available at [Kaba].

- *A process-aware IDS relying on temporal properties of the physical process*
  We design an IDS to detect attacks targeting the physical process controlled by the ICS. Model-wise, a physical process is composed of several variables that represent its state. The IDS we propose learns how the variables change over time and uses this information to detect process-oriented attacks. We compare our IDS with two other process aware IDS proposed in the literature on datasets from a real ICS testbed and a simulated ICS. The code for this work is available at [Kabc; Kabb].

Based on the above contributions, the thesis is organized as follows:

- Chapter 2 explains the context of this thesis. It gives an introduction to ICS, followed by a general discussion of the security of ICS and its challenges and our goals.
- Chapter 3 discusses IDS and their characteristics, with a focus on Flow-based IDS. It presents a tool to generate synthetic ICS network flow traces and uses them to evaluate the detection performance of different Flow-based IDS proposed in the literature.
- Chapter 4 first introduces flow whitelisting and SDN. Then, it presents two applications of SDN to implement security solutions for ICS: a two-level IDS and a countermeasure to eavesdropping attacks.
- Chapter 5 focuses on the detection of process-oriented attacks. It describes two popular families of approaches that have been proposed in the literature to detect such attacks and their limitations. It then presents a Time-based IDS and evaluates its performance on two datasets.
- Chapter 6 concludes this thesis. It provides an overall reflexion on our work and gives an outlook on the future of ICS security.

Bibliographic notes

Conference Publications

1. *A low-delay SDN-based countermeasure to eavesdropping attacks in industrial control systems, 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), main author*

2. *A Two-level Intrusion Detection System for Industrial Control System Networks using P4, International Symposium for ICS & SCADA Cyber Security Research 2018, main author*

Journal Publications

1. *Network trace generation for Flow-based IDS evaluation in control and automation systems, International Journal of Critical Infracstructure Protection, Volume 31, December 2020, main author*

Posters and Demos

1. *A Two-level Intrusion Detection System for Industrial Control system, Erasmus Mundus Joint Doctorate in Distributed Computing (EMJD-DC), 2017, main author*

Submitted and Under Review

1. *Exploiting the Temporal Behavior of State Transitions for Intrusion Detection in ICS/SCADA, IEEE Open Access, 2022, Submitted, main author*

Miscellaneous Contributions

1. HVAC Trace

Motivation and Background

2

This chapter explains the goal of this thesis and why the security of Industrial Control Systems (ICS) is important. It then gives an introduction to what ICS are, what their roles are, and how they operate. The chapter then discusses how ICS are different from traditional IT and what those differences mean for their security. It concludes by explaining the approaches that have been followed by researchers to detect intrusions in ICS.

2.1 Context and Motivation

ICS are computer systems used in industrial facilities such as water treatment facilities or power plants. Their role is to automate the monitoring and the control of physical processes occurring in the facility. Historically, ICS were isolated systems running in their own networks, but over the years operators have connected them to other networks, in particular corporate networks, to optimize business processes and decisions. To this end, existing IT networks technologies and protocols such as TCP/IP have been used to ease this interconnection. The result of this evolution is that many ICS are now directly or indirectly accessible from the outside and many of the vulnerabilities known from traditional IT networks can now be also found in ICS.

Consequently, the number of attacks targeting ICS has sharply increased in the past decade, although finding the exact number is challenging as companies are not always inclined to reveal information on the cyber attacks they have faced. Nonetheless, there exists a publicly available dataset from the Repository of Industrial Security Incidents (RISI) which contains security incidents related to ICS reported by companies during the period of 1982 to 2014 [Org15][Alr+18]. It currently includes 242 incidents of which 66% have been caused by cyber-attacks. Figure 2.1 shows the number of incidents per year for the period covered by the dataset. As it can be seen, over the decades, the number of reported incident has increased.

In 2020, IBM released a security report stating that the number of attacks targeting ICS facilities in 2019 was greater than what had been seen during 2016, 2017, and 2018 combined. They explained this phenomenon by the convergence of ICS and traditional IT systems, making it easier for an attacker to

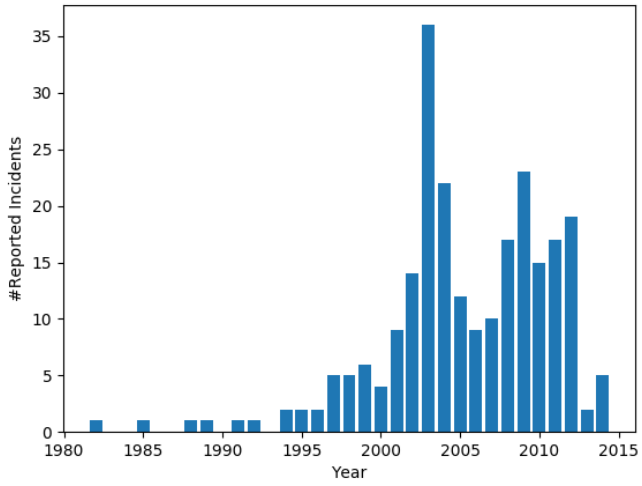


Figure 2.1: Reported incidents per year [Org15]

access the ICS by getting first into the IT system connected to it [Zei20]. In some cases, it is not even necessary to first breach into the IT system as some ICS components are directly facing the Internet, as shown by Shodan, a search engine that allows users to look for specific services (open ports) on the Internet. Shodan continuously sends probing packets to scan the Internet and collects information on hosts that have ICS-related services running. According to Anton *et al.* [Ant+17], more than 1.4 million ICS components accessible from the Internet could be found in that way in 2017. It is therefore not surprising that ICS have become an interesting target for cyberattacks. Some of these attacks became widely reported as they caused important damage. The most famous attack is arguably the one involving Stuxnet [HF+18], a sophisticated malware targeting the centrifuges of an Iranian uranium enrichment facility. By infecting the controllers of the centrifuges, the malware was able to increase the rotation speed of the machines such that they would break [Lan11].

Because of their role and their interaction with the physical world, the protection of ICS is an important matter. A successful attack can lead to economical damage, but human lives and the environment could be put in danger, too. However taking preemptive actions can be harmful as well. For example, the usage of an active Intrusion Prevention System (IPS), i.e., a system that detects attacks and autonomously takes countermeasures or mitigation actions, could have unwanted consequences if the detection capabilities of the IPS are not one hundred percent accurate. A more passive solution is the use

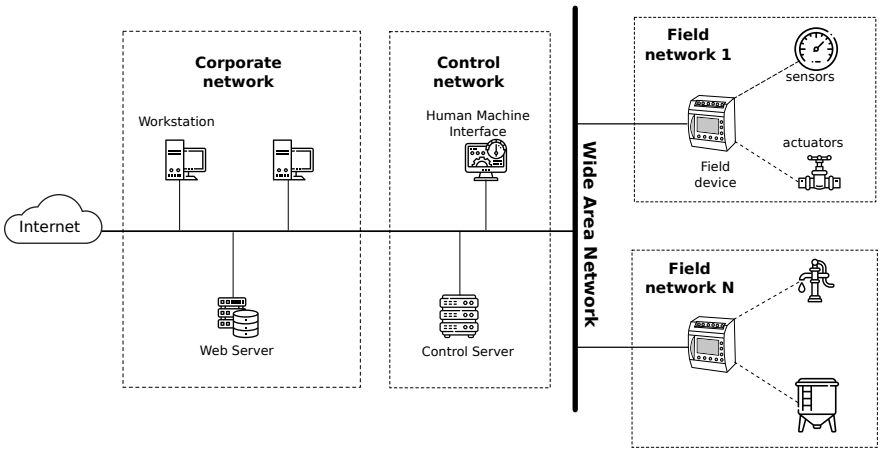


Figure 2.2: ICS architecture

of an Intrusion Detection System (IDS) which only detects attacks and leaves further defensive countermeasures to a human operator. While less efficient in terms of reaction to an attack, this approach avoids the situation where a false alert could result in devastating consequences. This is the approach that has been chosen by many researchers working on ICS security and which we also mostly follow in this thesis.

2.2 Basics of Industrial Control Systems

2.2.1 ICS Architecture

Although there are different types of ICS, such as Supervisory Control And Data Acquisition (SCADA) or Distributed Control Systems (DCS), the core idea is always to instrument and automate physical process operations. An ICS consists of different components that interact with each other to control and monitor a physical process. To allow such interactions, a communication network is required.

Figure 2.2 shows a typical ICS architecture of a company. It is logically split into several sub-networks, each having a specific role. The physical process monitored and controlled by an ICS can be scattered over thousands of square kilometers, e.g., a water distribution network, or be located within a small area, for example a factory. The devices in direct contact with the

physical process are organized in one or more *field networks*. The control and monitoring services of the ICS are centrally located in a *control network*. Finally, the *corporate network* hosts the office workstations and servers of the company. It is connected to the Internet and provides employees access to services such as e-mail. Often, access *from* the Internet to parts of the corporate network, e.g., to the company web server, is also possible.

The control network and the field networks are the core of the ICS and usually contain four types of components:

Control Server or Master Terminal Unit: The control server, also known as the Master Terminal Unit (MTU) is the brain of the ICS as it monitors all aspects of the physical process and takes decisions accordingly. It checks that all variables of the process are in acceptable ranges and initiates actions when it is not the case. The control server mainly communicates with the Human Machine Interface and the field devices to collect data and send commands.

Human Machine Interface (HMI): The HMI provides a graphical representation of the status of the physical process in real-time and allows operators to interact with the process. Using the HMI, operators can set control objectives or override automatic control operation through the control server.

Field Devices: Remote Terminal Units (RTU) and Programmable Logic Controllers (PLC) are small embedded systems that implement specific functions, such as I/O control, timing, and logic. They control actuators and monitor sensors and communicate with the control server to report measurements and receive commands. In order to increase fault tolerance, it is common to design field devices such that they operate safely even if the connection to the control server is lost.

Actuators and Sensor: Actuators and sensors are pieces of hardware that the ICS uses to interact with the physical world. They are connected to the field devices.

2.2.2 SCADA and DCS

There are two types of ICS, which differ not in term of their architecture but in how they operate. The first type is Supervisory Control and Data Acquisition (SCADA). SCADA systems are event-driven systems where the control server contact the field device to gather data and detect events. Once an event occurs, the system will trigger an action. The field devices are mainly use to collect data and perform simple commands sent by the server. SCADA are

also designed for long distance communications as they are meant to cover large geographical areas.

The second type is Distributed Control System (DCS). Those systems are more process-driven in the sense that more responsabilities are put on the field devices because DCS tend to control more complex processes. In DCS, control loops are used more on the field devices to automatically maintain set-points. DCS are used in more confined space than SCADA systems and communications are typically carried over a Local Area Networks.

There is an overlap between those two kind of systems and the differences are quite subtle as they can be easily blended. For the reminder of this thesis the term ICS will refer to as ICS/SCADA.

2.2.3 ICS Network Protocols: The Example of Modbus/TCP

As mentioned earlier, ICS have evolved in the past two decades from isolated systems using proprietary network technologies to inter-connected systems based on IP networks. Consequently, the network protocols have been also adapted. One of them is *Modbus*, first published by the company *Modicon* in 1979 and today one of the most popular protocols for the communication between field devices and control servers in ICS. It was originally designed for serial communication, but variants for other communication technologies have been created. Here we present Modbus/TCP, the variant of the protocol running on top of TCP/IP [Org06].

Modbus/TCP follows a client-server architecture where the Modbus/TCP client is the control server (e.g. the MTU) and the Modbus/TCP server is a field device (e.g. a PLC). In the Modbus/TCP data model, a Modbus/TCP server has different types of memory registers that can be queried or modified by the client by sending specific Modbus/TCP messages, as shown in Table 2.1. Discrete input registers and Input registers reflect the state of a physical input device attached to the field devices, for example a switch or a temperature sensor. Coil registers represent the state of an output port of the field device and can be used to control for example an attached relais. Finally, holding registers store configuration values that can be read or written by a client. In Modbus/TCP messages, the registers are identifed by a 16-bit address. The implementation of that data model and the mapping of the registers to the concrete hardware and I/O ports of the field device are left to the device manufacturer.

Modbus/TCP is an application-layer protocol in the TCP/IP model, therefore for a client and a server to interact, they need to establish a TCP session. Even though data transactions are usually stateless, as data becomes quickly obsolete because of the real-time nature of a physical process, the designers of Modbus/TCP have favored a connection-oriented transport protocol such

Name	Size	R/W
Discrete input	1-bit register	Read only
Coil	1-bit register	Read and Write
Input register	16-bit register	Read only
Holding Register	16-bit register	Read and Write

Table 2.1: Modbus/TCP Data Model

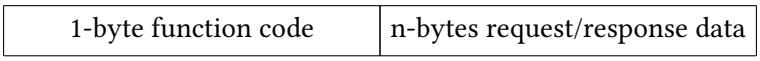


Figure 2.3: Modbus message format

as TCP over UDP due to the reliability the former provides without requiring specific actions from the Modbus/TCP applications. One of the reasons Modbus has become so popular is its simplicity as it only knows three types of messages:

- Request: Message sent by the client to query or modify registers on the server.
- Response: Message sent by the server in response to a request.
- Exception Response: Message sent by the server when an error occured during the processing of a request.

The messages follow the same basic schema, as shown in Figure 2.3. In a request, the *function code* specifies what kind of action the client wants the server to perform. The rest of the messages contains the data necessary to perform the action. The server performs the requested action and if the action was successful it replies with a response message with the same function code as the request message and additional data (see Figure 2.4). For example, a request with function code 2 ("read discrete inputs") contains the addresses of the Discrete Input registers to read, and the corresponding response contains the values of those registers.

Function codes are values between 1 and 127. Modbus splits function codes into three categories: public, user-defined, and reserved. Public function codes are publicly documented and validated. User-defined function codes are those function codes that are not supported by the specification but implemented by users of the protocol. These function codes can be made public after a thourough review by the MODBUS Organization. Finally, the reserved function codes are owned by companies for legacy products and are not available to the public. Table 2.2 shows the most used public codes.

Codes greater than 127 indicate an exception response. When a server is not able to perform a requested action, it will reply with a message where

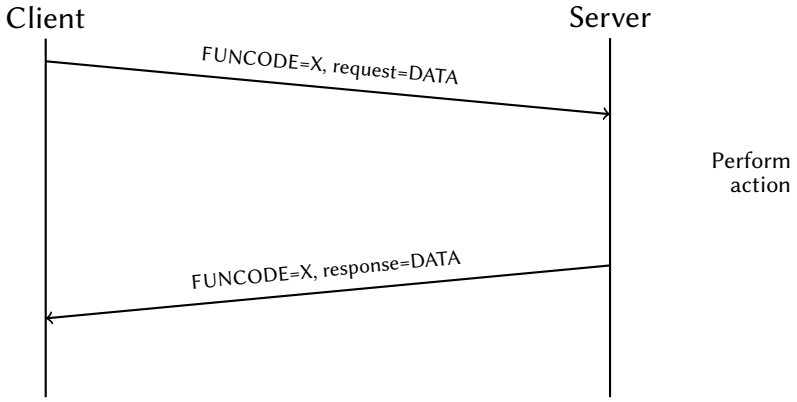


Figure 2.4: Modbus normal exchange

Function Code	Description
01	Read Coil
02	Read Discrete Input
03	Read Hodling Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Holding Register
15	Write Multiples Coils
16	Write Mutltiple Holding Registers

Table 2.2: Modbus/TCP data access function codes

1-byte exception function code	1-byte exception code

Figure 2.5: Modbus Exception Response message format

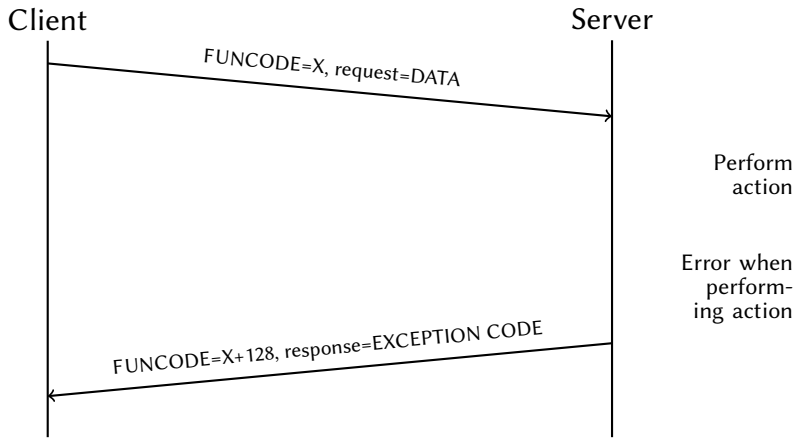


Figure 2.6: Modbus exchange with error

the function code is equal to the function code of the request incremented by 128. In that case, the data part of the message only contains one byte with the error code, for example code 1 ("Illegal Data Adress"), as shown in Figure 2.5 and 2.6.

The first Modbus implementation was for a serial line network where the message size was limited to 253 bytes. For legacy reasons, this limit was kept in Modbus/TCP. Additionally, a 7-byte Modbus/TCP header is prefixed to every Modbus message. Figure 2.7 shows the header. It contains four fields:

1. Transaction Identifier (TransID): This field is used by the client to identify each request/response pair. The server echoes the transaction identifier in its response, so that the client can match the response to the request.

2. Protocol Identifier (ProtID: This field is reserved for future extensions. For Modbus/TCP it is always set to zero.

3. Message Length (MsgLen): This field specifies the number of bytes that follow in the message.

4. Unit Identifier (UnitID): This field is used to identify a server in a non TCP/IP network. In TCP/IP networks, it is set to 0 or 255 and is simply echoed by the server.

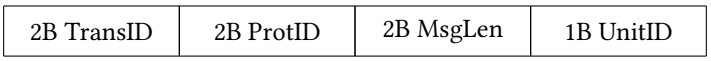


Figure 2.7: Modbus/TCP header

2.2.4 Operation Technology vs Information Technology

ICS are examples of *Operation Technology* (OT). The latter refers to any technology used to monitor and/or control physical processes through physical devices. In contrast, *Information Technology* (IT) refers to any kind of computer technology used to manipulate data or information. It includes many services, like mailing or databases, that most companies use for their business operation. OT and IT serve different purposes and for that reason they are used and managed differently [E+02]. Since a large part of our thesis is about network security for ICS, we will present in the following the major differences in the network communication between ICS and traditional IT networks such as the Internet.

First of all, because of the damage a faulty or misbehaving ICS system could cause, reliability is essential. For example, failing end hosts and network equipment are not acceptable in an ICS and in the rare case they should occur, the ICS must be able to still monitor and control the physical process. In the Internet, failures are undesirable but they are tolerated as they often "only" lead to loss of data. It is not rare to simply reboot a machine in order to fix it.

Another difference is how the network is used. Hosts in IT networks tend to exchange large amounts of data in short time caused by human activities. For that reason, IT networks have a relatively high througput compared to ICS networks. In addition, the traffic in many IT networks depicts patterns caused by human activities, for example a diurnal pattern [BSP13]. Humans being less active during the night, the utilization of the network and the number of active devices is lower during the night than during the day. In contrast, the field networks of an ICS consist of small embedded devices with limited computing and storage capabilities. They are not meant to generate or process a lot of data, but in order to work correctly, they might have hard real-time requirements on the network. As shown with Modbus, the monitoring of a physical process is often done in a polling fashion where the Control Server sends small requests to the field devices. The elapsed time between two requests is usually relatively long, e.g., one second. As a result, the network utilization is fairly stable and regular. Moreover, the set of active hosts stays mostly constant since devices run for long periods without interruption. Table 2.3 summarizes the differences.

	ICS	Internet
Failure tolerance	Low	Medium
Network utilization	Constant	Bursty
Volume data	Low	High
End-host addition/removal	Rare	Frequent

Table 2.3: Difference between OT and IT

2.3 Security of Industrial Control Systems

As explained, there exist several differences between OT and IT, and the fact that ICS got connected to IT networks and adopted their technologies did not get rid of these differences. In old ICS, the isolation was the only means of security but it got torn apart when that changed. Therefore, new security means are required for those systems, but because of the distinct characteristics of ICS compared to IT network, it is somewhat optimistic to think that the security measures that have been designed for IT networks could simply be ported to any ICS network. In addition to the difference between IT and OT, ICS have several constraints that hinder the design and use of existing IT security mechanisms.

2.3.1 Definition of Security

Before we continue it is important to understand what it means for an ICS to be secure. First of all, in this thesis, we are not discussing security that would prevent physical intrusions like an individual infiltrating a building. We are focusing on computer security, meaning the security of the hardware, the software, and the data that are part of the system. A well-established model to specify the objectives of cyber-security is the CIA model which stands for Confidentiality, Integrity and Availability. Those are the three properties that security measures should ensure:

- Confidentiality refers to the protection of data and the prevention of unauthorized access. No sensitive information should be retrieved from an unauthorized party. An example of a security measure tied to confidentiality is the use of mechanisms to encrypt data.
- Integrity deals with the reliability and the accuracy of data, for example ensuring that data have not been tampered. Again, cryptography can be used in this context, for example to generate signatures based on hash algorithms.
- Availability involves ensuring the constant availability of the information to authorized parties. A common attack to prevent access to data

(*denial of service*) consists in exhausting the resources used to provide the service. Increasing the resources and providing more redundancy are simple examples of measures to guarantee a better availability of data.

The above order of the properties reflects the decreasing level of priority generally given to each of them [NPP17]. This does not mean that any of the less important priorities should be ignored in favor of the more important ones, but it is an indication of what many companies want to prevent the most, which is the leakage of sensitive information. Being the victim of such a leakage can seriously tarnish the reputation of a company and negatively impact the trust of their clients towards them, especially in this day and age where privacy has become such an important issue.

However, for an ICS, the order of priorities is different. Availability is much more important, as a failure of any component could lead to catastrophic damages. Confidentiality, on the other hand, is less important as ICS data does not tend to hold very sensitive information and becomes quickly obsolete due to the real-time nature of the monitored physical process.

2.3.2 Vulnerabilities of ICS

In order to secure ICS, it is necessary to know the vulnerabilities that attackers can exploit to perform an attack. [SFS11] lists many vulnerabilities but the ones we are interested in here are related to the network, typically in the context of communication protocols and remote control access.

First of all, since the original ICS protocols were designed under the assumption that they would be used in isolated environments, their designers did not deem necessary to add security mechanisms, like encryption techniques or authentication [Pli+20]. This can be seen in Modbus/TCP which does not include such techniques. Because of that, an attacker who can connect to a Modbus/TCP server can basically do whatever they want, like writing to registers.

The second important vulnerability is control access. The fact that ICS got connected to other networks does not mean that anyone should be able to communicate with the hosts inside the ICS network. It seems like a simple fact, but ensuring that requires some management and configuration effort. It is not rare for the vendor of a component to have remote access to the deployed component for maintenance purposes, in this way increasing the attack surface. Additionally, if remote access is permitted to allow employees to work from anywhere, the ICS networks are more opened which makes them more vulnerable. Strict and clear control policies must be defined so that only authorized parties have access to the ICS, but that is not enough. The IT systems of the authorized parties themselves must also be secured to

avoid a situation where an attacker first compromises one of the parties to gain access to the ICS.

Another issue is the age of ICS equipment. Due to constraints that will be discussed in more detail in the next section, it is frequent for ICS to use outdated hardware for which vulnerabilities are known. A typical example are field devices which run for very long periods and cannot be easily updated or replaced.

2.3.3 Challenges

ICS are more stable than traditional IT systems. This means that changes to the software or hardware are rare and therefore it should be relatively easy, at least in comparison to IT networks with their large variety of services and protocols, to prevent that something happens that breaks the regular behavior of an ICS.

However, ICS have a lot of constraints that make their protection complicated. The first and main difficulty of ICS systems is the fact that they are difficult to modify and upgrade. This is due to multiple factors. First, as explained before, high availability is a very important property of ICS. This makes those systems and their components difficult to upgrade or update. As a consequence, it often happens that unpatched versions of software with known vulnerabilities are used in ICS. The second factor is the need for compliance. Software and hardware must be thoroughly tested and validated to ensure the absence of errors and bugs. Bugs can cause delay or even physical damage to the system. This process of testing and verification takes a lot of time and therefore modifying or upgrading an ICS is a slow operation. Additionally, no owner of an ICS wants to see information on their system published, therefore only a handful of people can perform such verification.

Another challenge is the fact that ICS are used to manage critical structures. For that reason, any solution to protect against attacks should be highly accurate. For example, a solution that blocks malicious packets must never block legitimate packets by accident. If such a false positive occurs, it can disrupt the normal operation of the whole system and corrupt the physical process.

The third challenge is the limitation of resources. Field devices used in ICS are small embedded systems that have limited resources and capabilities. Compared to IT network components, ICS components have a long lifetime, from 10 years to 20 years, which results in relatively old and outdated hardware. It is difficult to apply resource demanding security mechanisms such as encryption or anti-viruses.

The last challenge that should be mentioned is the mentality of ICS operators for whom safety is often more important than security. Old ICS were

considered as secure because no one could remotely access them. Although many existing ICS have serious security problems, operators fear that modifying them to improve security could have an impact on safety.

2.3.4 Attackers

Attackers of ICS can be classified into four categories [Do+17]. The first category surprisingly includes insiders of the ICS, meaning people who know well the system as they are or were part of it. They are the principal cause of sabotage of ICS, either accidentally or intentionally. An example is an unhappy employee who got fired and wanted to take revenge [SM08]. As insiders know the system well and potentially have access to it, they are highly dangerous from a security standpoint. The second category includes hackers who leverage the fact that ICS adopted IT software and protocols. Finding malicious scripts exploiting known vulnerabilities in IT and use them on an ICS is possible. There are even scripts specifically designed to attack ICS. The third category includes criminal groups or terrorists who target ICS in order to cause damage remotely without the need to come up with an attack plan that would require physical resources. The last category contains Nation-States that can use cyber-attacks against critical infrastructure as a weapon to enforce political views. The primary example of that category is Stuxnet.

2.4 Intrusion Detection Systems

Against the several threats faced by ICS, security mechanisms must be put in place. One of these mechanims is an Intrusion Detection System (IDS). As the name suggests, an Intrusion Detection System is a technology designed to detect intrusions occurring to a target system monitored by the IDS. The need for such a system rises from the fact that guaranteeing that a system is completely secure is challenging. Some vulnerabilities are unknown until they have been exploited by a zero-day attack. In that regard, even a detection mechanism that does not necessarily know how an intrusion has happened but is able to know that one has occured is an attractive solution, as it allows operators to investigate and react. Moreover, for multi-step attacks, detecting one of the steps before the attack is completed can prevent the attacker to achieve their final goal.

Several properties are expected from an IDS [Deb00]:

Accuracy: The IDS is accurate when it raises alerts during an attack and does not trigger false alarm. More precisely, it must correctly distinguish between malicious and legitimate activities.

Time Performance: The IDS is time efficient when it is able to detect attacks fast enough so that a security officer has the possibility to react to an attack. This property depends on the processing time of the IDS, that is how long it takes to analyze the activities occuring in the system, and the propagation time, i.e., how much time it requires to notify the security officer. If the processing time is too long, some activities may be missed by the IDS.

Completeness: The IDS is complete if it is able to detect all attacks. This property is hard to ensure as it is impossible to know all future attacks that can be performed against a system. However, there exist techniques to increase completeness.

Fault-Tolerance: An IDS is fault-tolerant if it is itself resistant to attacks and is able to to provide detection in any circumstance.

To measure the accuracy and completeness of an IDS, different metrics have been defined. Malicious activities that are correctly classified as such are denoted as True Positives (TP), while malicious activities misclassified as legitimate are denoted as False Negatives (FN). For legitimate activities, the ones that are correctly identified as legitimate are denoted as True Negatives (TN), while the ones wrongly classified as malicious are denoted as False Positives. With that, the Detection Rate (DR) is given by

$$DR = \frac{TP}{TP + FN}$$

which represents the ratio of activities correctly identified as malicious among all the malicious activities. The False Positive Rate (FPR) is given by

$$FPR = \frac{FP}{FP + TN}$$

which is the ratio of the number of legitimate activities that are misclassified as malicious among all the legitimate activities. Obviously, the goal is to have a high DR and a low FPR.

There exist many types of IDS that differ in how they detect intrusions, but they all follow an architecture composed of four parts. The first part is called the **knowledge base** of the IDS and represents a priori information that the IDS knows about the system that it monitors. The second part is the **data collector** which allows the IDS to interact with the target system to be aware of the current activities. The third part is the **detection engine** which is responsible for analyzing the data collected at the target system. It determines from those data and the knowledge base if there is an intrusion. The last part is the **response engine** which receives notifications from the detection engine when there is an intrusion and acts accordingly.

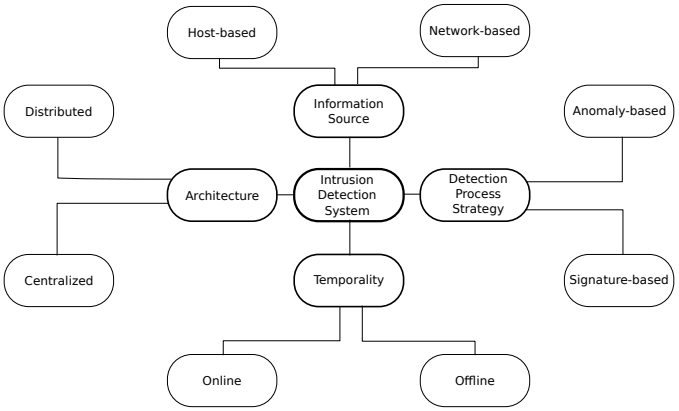


Figure 2.8: IDS taxonomy (adapted from [LKS05])

The ways those parts are implemented differ from one IDS to another. Researchers have attempted to classify the existing IDS according to different criteria. In the following, we will present a taxonomy proposed by Lazaveric *et al.* [LKS05] based on four dimensions: the source of information, the detection process strategy, the architecture, and the temporality (Figure 2.8).

.

2.4.1 Source of Information

An IDS can be classified according to the type of the target system and how data are collected to detect attacks. The target system can be a host in the network and in that case the IDS monitoring that host is referred to as a *Host-based IDS*. Or the target is the network and we call the IDS a *Network-based IDS*.

Host-based IDS (HIDS) are software running on a host (e.g. a workstation or server) that monitor the network communication, file system and login actions, etc. of the host. HIDS were the first IDS created, as the primary target of the first cyber-attacks were mainframe computers with users local to the system [Deb00]. An advantage of HIDS is that the interaction with the target system is more direct. However, since the IDS is running directly on the host, the overhead caused by the IDS can negatively impact the performance of the host. Another important drawback is that the information used by the HIDS is directly retrieved from the target system meaning that if a successful attack is able to modify this information (e.g., modify the logs) before the HIDS detects the intrusion, the attacker can hide their presence.

Compared to HIDS, Network-based IDS (NIDS) are not bound to a host but monitor the communications occurring in the network. They act as packet sniffers in the network that silently collect and analyze network packets.

NIDS that use Deep Packet Inspection (DPI) must understand the format of the packets and the data they are carrying. The increased usage of cryptography has rendered DPI-based NIDS less interesting. Other approaches, such as Flow-based NIDS, only rely on the non-encrypted meta data in the network traffic, such as port numbers and IP addresses. NIDS have to monitor an entire network. In a fast network where the network activity is high, i.e., packets are exchanged at a high rate, a NIDS may not have enough resources to process all the packets and therefore might drop some of them which would lead to a loss of information. The other important element to take into account when using a NIDS is the placement in the network as it impacts the traffic that the NIDS will be able to monitor.

2.4.2 Detection Process Strategy

IDS use different approaches to perform the detection on the collected data. A *Signature-based IDS* relies on a set of patterns ("signatures") created from the attacks expected to be performed against the target system. During the detection phase, the IDS monitors the activities of the target system and when they match one of the known attack patterns, an alert is raised. As an example, imagine the situation where the network activity of a host is as follows: it receives many TCP SYN packets but no responses to the SYN/ACK packet that it sends back. This is an indication that a SYN Flood attack is happening, so if there is a pattern describing SYN Flood attacks available to the IDS, it will raise an alert when observing such network activities. The main drawback of Signature-based IDS is that its performance is directly tied to the set of available attack signatures. Attacks for which the signature is not known to the IDS will be missed. The main advantage however is that these IDS generally raise few false alerts and that they cannot only detect but also identify the attacks.

In an *Anomaly-based IDS*, the knowledge base contains information about the normal, expected behavior of the target system. The IDS looks for any activity that deviates from that behavior. An example for that infomrmation can be a model stating that users log on their computer every business day for two hours. If the IDS observes a deviation in the behavior of the system from the model, it raises an alert. The advantage of Anomaly-based IDS is that they can even detect new attacks which makes them more complete than Signature-based IDS. But Anomaly-based IDS suffer from two weaknesses. First, they require an accurate description of the expected behavior of the system. In many publications, machine learning is used to automatically learn the behavior model from past observations during a learning phase, which makes the duration of the learning phase important. If it is too short, the IDS might fail to include rare normal events in the model and will treat them later,

during the actual monitoring phase, as anomalous. Moreover, if an attack occurs while the IDS is in the learning phase, the attack will not be detected during the monitoring phase as it will be considered normal.

2.4.3 Architecture and Time Strategy

The two last criteria to categorize IDS are their architecture and the time strategy. The architecture refers to whether or not the IDS is distributed or centralized. A distributed IDS has several independent agents analyzing data to detect an attack. The number of agents can vary during the lifetime of the IDS. While independent, these agents can exchange information with each other to detect more complex attacks that involve multiple systems. Centralized IDS can also consist of several components but a fixed number of them perform the data analysis in a tightly coupled way.

The time strategy describes when the IDS attempts to detect attacks. It can either be online or offline. An online IDS analyzes the data collected from the target system in real-time or near real-time, while and offline IDS detects attacks by doing a post-analysis of the data. An online IDS has the advantage of detecting attacks sooner thus allowing operators to react faster, but it requires that the IDS has enough processing power to keep up with the continuous flow of data from the data sources. This can be a problem during attacks that generate a lot of data, such a Distributed Denial of Service attack. An offline IDS detects attacks only when the post-analysis phase starts. Therefore, an attack that could have been detected by an online IDS, can remain undetected for a long period of time. However, the analysis can be more complex as there is no time constraint.

2.5 Intrusion Detection Systems for Industrial Control Systems

With the rise of cyber-attacks against Industrial Control Systems, researchers have decided to develop solutions to detect them. Non-intrusive passive Network-based IDS have been preferred in ICS environments for two main reasons. First and foremost, they do not require modifications on the existing servers and field devices, which is important with regard to the availability requirement of ICS and the limited resources of the field devices. Second, the passive nature of an IDS means that in the case of a false positive no potentially damaging countermeasure is automatically taking by the system.

Although signature-based IDS exist for ICS, we are interest in Anomaly-based detection in this thesis due to its advantage described in Section 2.4.2.

2.5.1 Anomaly-based IDS for ICS

As it has been explained, ICS networks are much more stable in terms of topology and communication. This stability makes the behavior of the overall system more predictable which is why many IDS that have been proposed by researchers are Anomaly-based IDS. Since the system is more predictable, building an accurate model of the "normal" network communication dynamics and behavior is expected to be easier than in more general communication networks. In the following, we give some examples of existing Anomaly-based IDS to illustrate the wide range of detection methods found in the literature.

Yang *et al.* [YUH06] build the model of the normal behavior through statistical techniques. The authors first create multiple traffic profiles from several sequences of measurements over a specific period of time. The measurements include network features collected via the Simple Network Management Protocol (SNMP) and hardware features such as CPU usage or link utilization. Traffic profiles are then classified according to when the measurement was carried out. The profiles are fed to an *Auto-Associative Kernel Regression* (AAKR) to build a model that will predict the valid behavior. During the monitoring phase, a new profile will be created and given to the AAKR model. The model compares the profile with what is expected and generates residuals which represent the difference between the two profiles, actual and expected. These residuals are then submitted to a binary test called *Sequential probability ratio test* that decides whether the new observation is suspicious or not.

Gao *et al.* [Gao+10] use a neural network to create the model. The neural network is trained with captured packets containing requests and responses exchanged between the control server and the field devices. The neural network uses as input the content of the messages and the frequency of the request/response transactions. This kind of model is able to detect attacks that inject additional requests and responses with the goal to provoke a denial of service.

Goldenberg and Wool [GW13] use Deterministic Finite Automaton (DFA) to model the communication between a field device and the control server or the HMI. It not only covers the function codes but also the addresses of registers and coils that the client wants to access. These characteristics will be used to create the alphabet of the DFA and the transition function. The states are the positions of the messages exchanged on the channel for a period of time. For example, if every 30ms two queries are sent to a PLC, there will be four states for the DFA modeling this channel. Inputs that are unknown to the IDS or are not at the position they should be in the DFA trigger an alert.

More recently, Lin *et al.* [LNA17] proposed an IDS that bases its detection

on the timing of the communications occurring between hosts. During the learning phase, the IDS extracts a sample of size W of inter-arrival times for every event of interest (e.g. packets received/sent) and estimates the mean, the standard deviation, and the maximum and minimum of the inter-arrival times in the sample. With those values, the IDS compute an upper threshold and a lower threshold. During the monitoring phase, a sliding window is used to compute the current mean, standard deviation, and range. If the observed values are outside the learned thresholds, an alert is raised.

Cheung *et al.* [Che+07] proposes three approaches, of which the first two are specification based. In the first one, the Modbus specifications are used to create the model. Those specifications describe the expected communication behavior in different ways. One way is to look at the individual fields defined in the protocol. Typically, for the Modbus/TCP protocol, listing the function codes used by a PLC is the first step to model its behavior. Specifications can go further by describing the dependency between multiple fields. An example of dependent fields are the function code and the message length. If a specific function code comes always with a specific message length, we can include this information in the model. Another way is to look at the combination of request and response messages. Some fields in the response must match their counterparts in the request. Furthermore, some requests are sent periodically and the corresponding responses are almost always the same, so they can easily be predicted. The second approach models the communication pattern between components of the network. This results in a list of network access policies. If a communication violates a policy then an alert is raised. The last approach focuses on the availability of the servers and services. The system first learns the services available in the network it monitors. Connection attempts to services unknown to the system are considered suspicious. This approach is more useful in IT networks which provide many services but it can be adapted to ICS networks by considering the fact that the services in an IT network are kind of equivalent to the function codes supported by a PLC.

Fovino *et al.* [Fov+10] follow an approach that is very different from the above ones. They describe the critical states of the system via a language they have created. A critical state is defined as a state or configuration that can put the system at risk. The system is divided into subsystems. Each subsystem is monitored by a local IDS which will periodically send a request to the component of the subsystem to gain information about their status. Each local IDS is connected to a global IDS which will collect the critical state alerts from the local IDS and infer global critical states.

2.5.2 Flow-based IDS for ICS

Many of the work presented above rely on information that the IDS collects by using Deep Packet Inspection (DPI), i.e., by inspecting the payload of the packets traveling through the monitored ICS network.

Even though many existing ICS do not use encryption, DPI is still problematic. ICS manufacturers often use proprietary protocols or protocol extensions, which means that a DPI-based IDS must be manually adapted to understand the payload. Moreover, it can be expected that future ICS will use encrypted protocols [KM+08; McK+17]. With that premise, some researchers have proposed solutions using Flow-based IDS [BSP13; ZR17; KČD12].

Flow-based IDS see network traffic as a set of flows, where a flow is defined as a sequence of packets with common characteristics. For example, a flow can be defined as all packets with identical source and destination addresses. Only aggregated metrics of the flows (such as the total byte size) are fed to the IDS. Information about individual packets, especially their payload, is not collected. Flow-based IDS have been applied for several years to Internet traffic [So +10]. Compared to DPI-based IDS, Flow-based IDS are more limited in the types of attacks they can detect as they do not inspect the payload. However, they are less protocol specific. Visibly, there is a tradeoff between the detecting capabilities of the IDS and the range of systems that it can handle.

2.6 Conclusion

In this chapter, we have discussed the importance of securing ICS against cyber attacks and the challenges that come with performing such a task. We have also presented the different techniques that IDS use to detect attacks and we have explained why we are interested in Anomaly-based and Flow-based IDS.

One of the main challenges that we have seen is that making modifications on the end hosts of ICS to remove all vulnerabilities, especially on the field devices, is complicated. For this reason, in this thesis, we focus on ways to mitigate the exploitation of those vulnerabilities. We focus on IDS and other security mechanisms that do not require such modifications, in particular on Network-based IDS. To achieve this goal, we look at three different directions that are formulated below as questions:

1. Traditional IT networks and ICS networks are different. Can the differences between the two be an obstacle when using Flow-based IDS from IT in OT?

2. What defense mechanisms can a Software-Defined-Networking enabled ICS use to defend itself against network oriented attacks?

3. Assuming that an attacker can tamper with the physical process through the network, how to detect their actions or their consequences?

The first question originates from fact that it may not be necessary to reinvent the wheel in order to secure ICS systems. Given that those systems now use traditional IT technologies, it would be interesting to see if security techniques of IT networks can be applied to ICS networks despite the differences between the two.

Software-Defined Networking (SDN) is a new networking paradigm which allows an easier management of the network. With the interconnection of the different types of networks (corporate, control and field network), access control and network segmentation are two important mechanisms to prevent unwanted connections between hosts inside the same network and from one network to another. With a better management of the network and the ability to program it, SDN provides interesting features to enforce network segmentation and access control.

Finally, regarding the third question, an attacker may target the physical process itself and not the ICS equipment. Such an attack can be performed through the network by sending commands to field devices so that they take specific actions leading to the disruption of the physical process. We want to detect when this happens so that the operator can react.

We provide answers to these questions in more detail in the next chapters of this thesis.

Flow-based Intrusion Detection Systems for ICS

3

In the previous chapter we discussed Flow-based IDS and their relevance for ICS networks. Several Flow-based IDS can be found in the literature, some of them designed for general IT networks, some of them specifically for ICS. Unfortunately, evaluating and comparing their performance in real and operational ICS is nearly impossible since ICS operators are, for obvious reasons, very reluctant to give researchers access to their systems. Instead, when evaluating new IDS algorithms, most researchers rely either on experiments in testbeds and simulators or use traffic traces collected in such environments for offline experiments [18; LF16; Rin+17]. The advantages of those approaches are the reproducibility of experiments, the flexibility to test different configurations, and the fact that no real system is put in danger. However, they require that the used environments accurately reflect the behavior of a real system, which is challenging to ensure. Publications relying on empirical data from *real* systems are much rarer and those few that exist have not published their datasets [KČD12; ZR17].

In this chapter, we present a tool to generate network traces for the evaluation of Flow-based IDS. The generation is based on statistical properties that the tool extracts from empirical traces. We demonstrate its usability by applying it to an empirical trace that we have collected at the Building Management System of a university campus.

We first present the empirical data trace in Section 3.1. We have made the trace available to other researchers, as we consider it as valuable on its own. It allows researchers to compare their results as well as to study and better understand the characteristics of automation systems and their differences to IT networks.

We explain our approach to generate new network traces containing legitimate traffic as well as malicious traffic in Section 3.2. Being able to produce new traces allows evaluating how an IDS would perform in different situations or in the presence of unexpected side-effects. Our approach takes as input a traffic trace from which it extracts the parameters for the generation process of the legitimate traffic. For malicious traffic, attacks are chosen from a pool of attacks described in the literature.

In Section 3.3, we use the traces generated with our tool to evaluate sev-

eral Flow-based IDS proposed and published by other researchers.

3.1 A Public Building Management Network Trace

A Building Management System (BMS) is a type of automation system used to manage different aspects of a building like control access or the Heating, Ventilation and Air Conditioning system (HVAC). In the following, we present and describe a public BMS dataset collected from part of the HVAC system of a university campus managed by Honeywell. The dataset contains the headers of the network packets exchanged between field devices, control servers, and Human Machine Interfaces (HMI) of the BMS. The dataset does not contain the payload of the packets due to privacy and security reasons, but it can be used for purposes where traffic traces on packet header level or flow level are needed, such as

- flow-level traffic characterization,
- parametrization of traffic models for synthetic traffic generation,
- experimentation with realistic background traffic load.

The system is fully automated, with a server communicating with peripheral devices deployed on the campus to control heating in offices, classrooms, and lecture halls. The part of the system that we have monitored controls around 15 to 20 buildings. It has been optimized for energy efficiency: When a teacher books a classroom or lecture hall via the booking application, a message is sent to the main server so that it can instruct the field devices responsible for that room to activate the heating several minutes before people are coming in. Operators access the system through the Human Machine Interface (HMI), which consists of dedicated workstations with a graphical user interface, to see in real-time the current system state or to set temperature goals manually.

3.1.1 System Overview

The BMS, topology-wise, follows a typical design for automation systems [SFS11]. A communication network interconnects the control server, the HMI stations, and the peripheral devices. The sensors and actuators are attached to the latter. Therefore, the peripheral devices are similar to PLCs and RTUs in SCADA systems: They collect and send sensor data to the server and their behavior can be (manually) influenced by the HMIs, but most of the time they execute simple control programs with setpoints defined by the server. The network is isolated from the rest of the campus network. The communication protocols are proprietary and use TCP/IP as the transport layer.

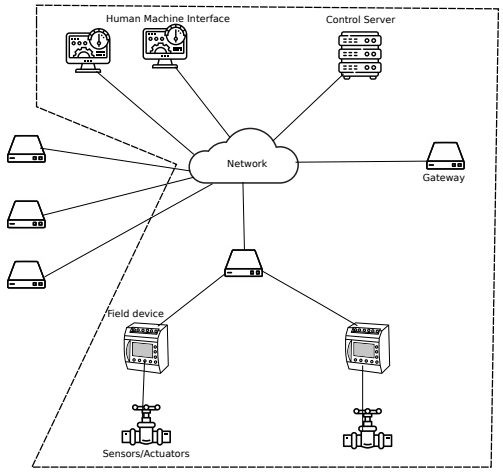


Figure 3.1: Network overview

It is important to note that the peripheral devices are not directly connected to the network. Instead, a group of devices is connected to a gateway that acts as an endpoint for the TCP connection from the control server (Figure 3.1). This allows the use of devices that are not IP-capable. In the part of the network that we consider for our dataset, there are 8 such gateways, each one connected to 1 to 15 peripheral devices. There is also a remote connection using a Virtual Private Network to allow remote access.

3.1.2 Packet Collection

In order to collect all network communication in the BMS network, we capture packets at two routers via port mirroring. The usage of two vantage points is necessary because the paths between the server and the peripheral devices are asymmetric.

We use *tcpdump* to capture the packets. For each packet, *tcpdump* records a timestamp of when the packet was captured based on the clock of the capturing device. Since we use two devices, we synchronize their clocks with the Network Time Protocol (NTP) so that packets appear later in the correct order when merging the individual packet traces based on the timestamps. Unfortunately, it showed in our first test runs that this clock synchronization is not sufficient and time-shifts ranging from 3ms to 10ms can be observed, resulting in wrong packet orders.

To obtain a more accurate timing, we periodically compute the shift between the two clocks and correct the recorded timestamps accordingly when merging the traces. To this end, capture device A sends every second an ICMP echo request to device B which will reply with an ICMP echo reply. These

ICMP packets appear in the traces of both devices. The current time shift s between the two clocks is then estimated by

$$s = t_B - t_A - RTT/2$$

where t_i is the timestamp of the ICMP echo request as seen by device i. The round-trip time RTT between the two devices is estimated as $RTT = t'_A - t_A$ where t'_A is the timestamp of the ICMP echo reply as seen by device A. Computing the network delay between the two devices in this way assumes that the delay is identical in both directions.

Using this approach, the number of out-of-order packets in a trace of one hour duration can be reduced from 8500 to 85. By increasing the frequency of the ICMP packets, we could further decrease this number but we do not want to send too many packets and add artificial load to the network.

3.1.3 Merging and Anonymization

When merging the packet traces from the two capture devices, we calculate for every ICMP echo request the time shift s using the method described above. We then adjust the timestamps of all packets captured by device A in the second following the ICMP request, i.e. the time until the next ICMP request. In the end, we obtain merged and time-corrected traces containing the packets captured by the two devices. In a filtering step, we only keep packets that either originate from or are destined to a host being part of the HVAC system, typically the control server, an HMI, or a gateway. We remove packets from protocols that are purely related to network management, namely ARP, STP, VRRP, LLMR. The original traces also contain SNMP traffic sent by the workstations to several devices inside and outside the network. Unfortunately, only the SNMP requests are visible to our capture devices so we remove them. We publish the traces in form of pcap files where each pcap file contains one hour of traffic.

Before publishing the traces, we discussed with the parties that would be affected by having the traces publicly available, following the advice in [DBK13]. These parties included the Chief Information Security Officer of the university the dataset was collected at and Honeywell, the company that deployed the HVAC system and who are the manufacturers of the devices, and the authors of the protocols used by them. We got their agreement after we explained what we wanted to publish and why.

During the processing of the traces, we made sure that no personal data was collected and that the traces can not be used to infer the behavior of individual people. The dataset only contains network traffic generated automatically, meaning that it is not the direct result of human action, except

for some part of the communication between the HMI workstations and the control server.

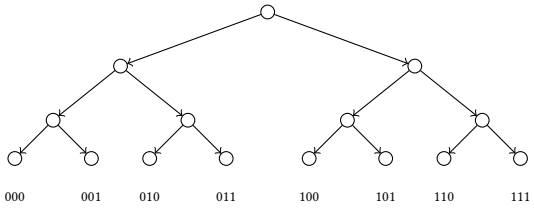
The owner of the BMS has demanded that the dataset is anonymized before publication. To do so, several steps were taken. First, we use CryptoPan [Fan+04] to anonymize the IP addresses in a prefix-preserving manner. CryptoPan uses cryptographic techniques to anonymized IP addresses while ensuring that the subnet structure existing within the original IP addresses are kept with the anonymized version. Additionally, it is being consistent across several traces given that the same key is used for the anonymization. It does so by representing the set of IP addresses in a binary tree with a height equal to the length bit of an address (e.g 32 for IPv4) where each leaf is an address. Excluding the root, each node of the tree corresponds to a bit value whose position is given by its height in the tree and the value whether it is in the left or right branch of its parent. During the anonymization process, CryptoPan will assign to each non-leaf node, including the root, a variable stating whether or not this bit has to be flipped. The function used to make the assignment is pseudo-random and seeded with an encryption key. The algorithm will then XOR the non-leaf node of both trees to get the anonymyzed addresses. Figure 3.2 provides a illustrative example of this process. To complete the anonymization, we then truncate the packet payload to only keep the headers and zero out the Organizationally Unique Identifier (OUI) of the MAC address.

3.1.4 Traffic Characteristics

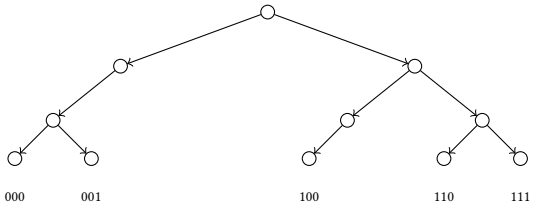
Now that we have explained the systems of interest and explained the collection process, we can present the main characteristics of the collected traces. We will discuss some of these characteristics in terms of unidirectional flows. We define a flow using the usual five-tuple (source IP address, destination IP address, source port, destination port, transport protocol). When we refer to flow size or packet size in the following, we always mean the size of the payload, excluding headers. For TCP flows, we only consider the ones where at least one byte of the payload was exchanged, thus removing connections that failed to be established. Note that our flow definition ignores TCP flags (SYN, FIN, etc.), i.e. a flow more or less corresponds to the definition of a flow record in Netflow/IPFIX [CTA30] with active and inactive timeout set to infinity.

3.1.4.1 Overview

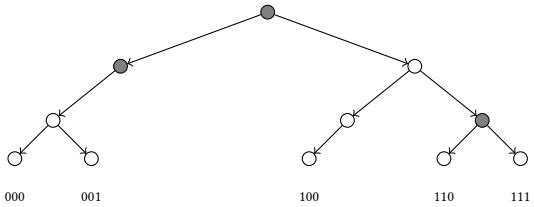
We collected the traces over a period of one week (7 days) in order to capture the behavior of the BMS on weekdays as well as during the weekend. Table 3.1 shows a summary of the part of the HVAC system we have monitored. The right three columns give basic statistics of the bidirectional communication



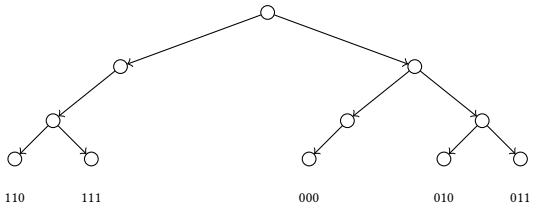
(a) 3-bits address space



(b) Set of original address



(c) Anomization process (gray nodes are flipping bits



(d) Set of anonymized address

Figure 3.2: CryptoPan anomyzation for 3-bit addresses

between HMI↔server and server↔gateways, respectively. The peripheral devices are not IP-capable, so we could not capture the traffic between them and the gateways.

Host Type	Number	Communicate with	Payload exchanged	Nbr pkts exchanged
HMI stations	7	Control Server	6.7GB	24.7M
Control Server	1	Gateways	387.7MB	11M
Gateways	8	Periph. devices	–	–
Periph. devices	58	–	–	–

Table 3.1: HVAC summary

In total, we collected 11.2 GB of packet data (corresponding to 4GB of packet headers) from 47 IP addresses and approximately 23,000 flows. Table 3.2 shows some characteristics of the entire dataset and the flows. As expected of an automation network, flows are small, the largest one transported 614.7 MBytes over one week.

dataset		flows		
			bytes	packets
size	11.2GB	min	5B	1
duration	7 days	avg	342.2kB	1725.08
Nbr flows	23K	max	614.7MB	5.1M
Nbr IP	47			

Table 3.2: Trace Summary

Figure 3.3 shows the empirical CDF of (a) the number of packets per flow, (b) the number of bytes per flow, and (c) the flow duration. The CDFs show that there is a significant number of flows with a very long duration and a large number of packets, although the distribution of the number of bytes per flow is rather conservative. Such a behavior is typical for automation systems where the server and the peripheral devices periodically exchange small control packets through more or less permanent TCP connections. In fact, all those connections in our trace lasted over the entire measurement period.

3.1.4.2 Application Protocols

Most of the observed traffic is from communication between the HMI stations and the control server. The HMI stations and the control server communicate through several ports. First, they exchange Remote Procedure Calls for Distributed Computing Environment (DCE/RPC) packet [KHT95]. The control

server acts as the DCE/RPC server on port 135. Surprisingly, an inspection of the packets revealed that other ports were used as well to exchange DCE/RPC packets. Those ports were not fixed; they changed during the duration of the capture. The protocol used for the communication between the server and the gateways is proprietary (port 2499).

In addition to the protocols used between the different components of the HVAC system, we observed other protocols. These protocols include ones from the NetBIOS family (NBDS, NBNS, NBSS) and are used by the HMI workstations and the control server to exchange data with each other and discover available services in the network.

We also observed some S7 communication, a protocol used by Siemens PLCs, field device. This traffic was related to another part of the BMS but we kept it in the dataset as it demonstrates an example of heterogeneity in automation systems. Web traffic was also observed, we discuss it in Section 3.1.4.5.

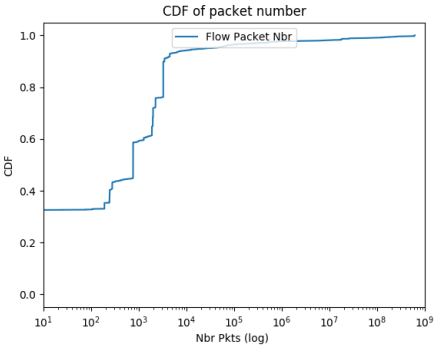
3.1.4.3 Network Activity Time Series

As mentioned in Section 3.1.1, the peripheral devices are connected to the IP network through gateways. The various devices differ in age and capabilities and therefore depict different communication characteristics. We illustrate this in the following with two gateways X and Y which connect 11, resp. 13, peripheral devices. In the part of the BMS monitored by us, 6 gateways have the same profile as gateway X and 2 gateways are similar to gateway Y, but with different numbers of peripheral devices connected to them.

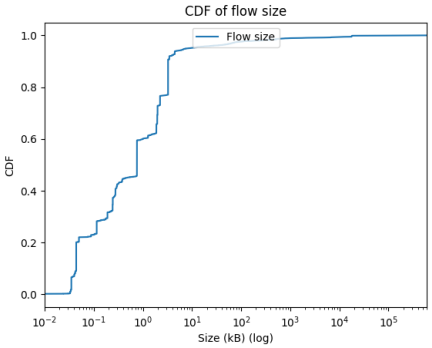
We plot the time series of the number of packets and number of bytes exchanged per hour between the main server and the gateways. Figures 3.4a and 3.4b (resp. 3.5a and 3.5b) show these plots for gateway X (resp. Y). We note some differences between the two gateways. First, despite having a comparable number of devices attached to them, the number of packets observed at gateway Y is considerably higher than at gateway X. A similar observation can be made for the number of exchanged bytes. We also notice how differently the packets are generated in both directions. In gateway X, the number of packets sent to the server is close to the number of packets sent in the other direction. In fact, most of the packets sent by the server to gateway X are ACK segments. We can see that the server behaves differently in the case of gateway Y: It actively sends segments with control commands to the gateway. Since the gateways are connected to devices of different models, it means that the devices connected at gateway Y are more data-intensive.

Finally, we observe diurnal and weekly patterns in the timeseries. The measurement was started on a Friday at around 2 PM, therefore the first 55 hours of low activity in the timeseries correspond to the weekend. This be-

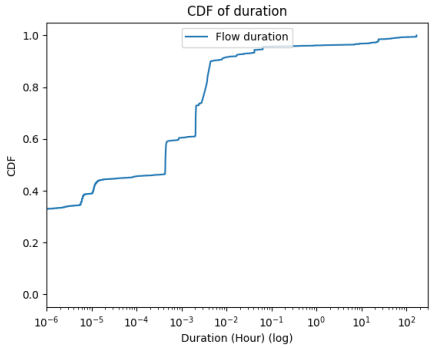
havior is understandable when considering the nature of the physical processes controlled by the BMS, but as mentioned when discussing the difference between OT and IT, it is rarely the case [BSP12b]. Interestingly, gateway *Y* shows lower activity during the night than during the weekend although the buildings are not used in both situations. We were not able to get the reason for that behavior but our assumption is that there is a kind of sleep mode enabled during the night which decreases the frequency at which data are exchanged.



(a) CDF packets per flow



(b) CDF kBytes per flow

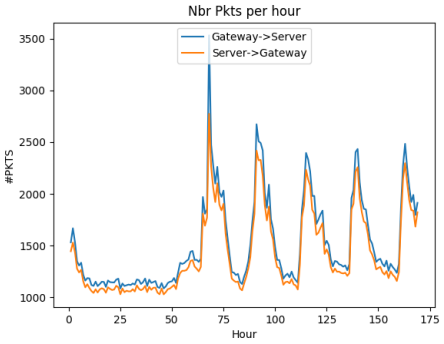


(c) CDF flow duration (in hours)

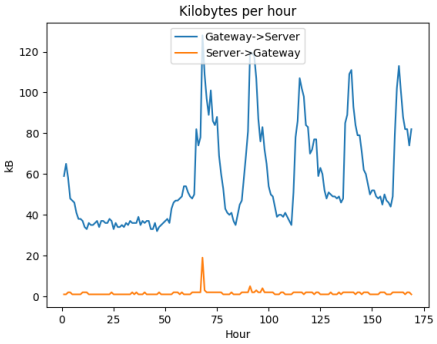
Figure 3.3: Flow characteristics

3.1.4.4 Packet Size Distribution

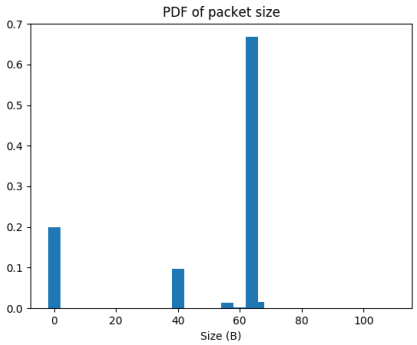
SCADA network protocols such as Modbus/TCP [Org06] or DNP3 require that the payload fits in a single IP packet to avoid packet fragmentation. These protocols aim at low latencies or even real-time usage, so packet fragmentation is undesired. We study the packet sizes for the entirety of the packets sent



(a) Packets per hour



(b) KBytes per hour



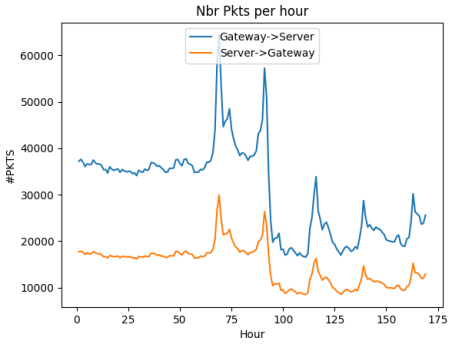
(c) Packet size distribution

Figure 3.4: Traffic observed at gateway X

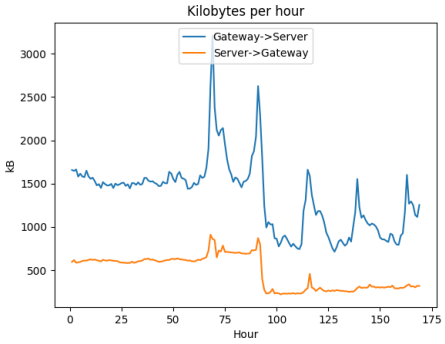
from both gateways to the server. The gateway X sent 255,251 packets and the gateway Y sent 5,183,021 packets. Figure 3.4c and 3.5c show the normalized histograms of the observed packet payload sizes in bytes for each gateway. Similar to SCADA systems, the packet sizes are rather small in both cases — they do not exceed 170 bytes. Moreover, only a few different sizes can be observed. Similar to other characteristics, there is a difference between the two gateways. Most of the packets sent by the gateway X contain actual data and there is only a small percentage (approx. 20%) of empty ACK while gateway Y sends more than 2 million empty packets.

3.1.4.5 Flow Stability

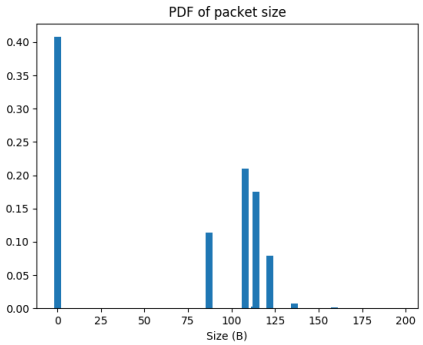
As mentioned previously, automation networks such as ICS are more stable in terms of traffic activities, such as the number of TCP connections established. This is also mostly true for the BMS network that we studied.



(a) Packets per hour



(b) KBytes per hour



(c) Packet size distribution

Figure 3.5: Traffic observed at gateway Y

We show in Figure 3.6 the number of new flows discovered per hour. For this figure, we have counted flows directly related to the HVAC system, i.e. flows between the control server, the HMI workstations, and the gateways, separately from other UDP and TCP flows. As it can be seen in the figure, except for the first hour when we start the capture, the number of flows discovered for the HVAC system is mostly constant. Actually, flows from the gateway to the control server lasted for the whole duration of the capture and there was no new flow created. The peaks that can be seen on the HVAC result from the change of the port used by the control server for DCE/RPC. The number of TCP flows discovered usually does not exceed 500 except at some point where we can see a high peak. The TCP and UDP flow, apart from the HVAC system, are mostly HTTP/HTTPS and DNS flows created when the HMI stations check if there are new updates. The peaks are caused by actual software updates that took place during the measurement period. We have

deliberately kept these "anomalies" in the dataset as they illustrate that even stable BMS systems can show irregular traffic patterns which might confuse intrusion detection systems.

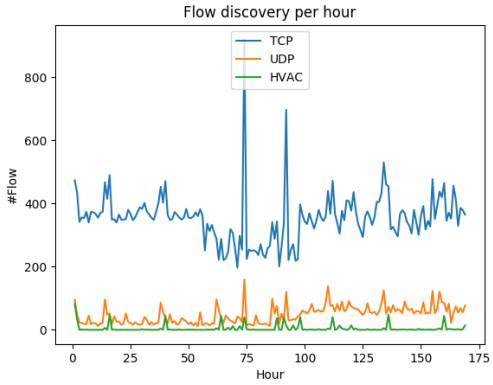


Figure 3.6: Flows seen per hour

3.1.4.6 Related Work

The characteristics of network traffic in industrial control and automation systems have been discussed in several publications. The SCADA network traffic of different utility companies is studied in [BSP12b; BSP12a]. In [KČD12] and [ZR17], the authors analyze the network traffic of BACnet-based BMS. To our knowledge, none of the datasets used in those and similar publications is publicly available due to privacy and security concerns. Therefore, researchers without access to a real system have to rely on public datasets collected in testbeds and virtual environments [18; LF16] or set up their own infrastructure for experiments, for example in [Lin+13].

As explained, our dataset does not contain payload data nor malicious traffic. Nevertheless, it can be used in research on Flow-based network intrusion detection systems that do not rely on deep packet analysis. With tools like tcpreplay [Kla30], researchers can replay the traces to obtain (benign) traffic that can be mixed with synthetic attack traffic for their experiments. Moreover, the traces contain valuable packet-level information for Anomaly-based intrusion detection [Gar+09], such as the timestamp and size of each packet. This data allows studying the dynamics of the HVAC flows which is useful to create a model of the system. The traces can also serve as a basis for trace generation, as we will demonstrate in the next sections.

3.2 Trace Generation for Flow-based IDS Evaluation

3.2.1 Motivation

Because of the differences between IT and ICS, existing IDS solutions from IT environments have to be carefully tested against ICS datasets to see if there are still valid in that context. Likewise, new IDS designs for ICS networks have to be evaluated. Many public network datasets exist for IT networks [20a], but to our knowledge, nothing comparable exists so far for ICS, which is a major problem for researchers working on ICS security. This highlights the need to generate ICS datasets. Tools to generate traffic traces exist for IT networks. However, we argue in the discussion of related work (Section 3.2.2) and in the presentation of our results (Section 3.3) that our dedicated approach is more suitable to ICS than tools designed for the generation of Internet-like traffic.

We identify three requirements to a trace generator for the evaluation of Flow-based IDS:

1. **Accuracy**: The generated traces must replicate the behavior of real ICS at flow-level. In other words, the flow traffic patterns in the generated traces must be similar to the ones one could expect from a real ICS. As we will explain in Section 3.2.3.1, our trace generator replicates bidirectional flow-level characteristics as well as certain packet-level characteristics. Although the latter are not strictly needed by pure Flow-based IDS, their correct emulation increases the flexibility of our trace generator because it does not impose a unique flow definition on its users. However, we do not require a perfect replication of packet header fields and signaling packets, such as ACK packets.

2. **Diversity**: The generated traces must contain benign and malicious traffic. The synthetic traces are generated in order to evaluate Flow-based IDS, in other words, their purpose is to assess the ability of a Flow-based IDS to make the distinction between legitimate and malicious traffic. Therefore, these two kinds of traffic must be included in the generated traces. In real-world scenarios, targets of attack traffic will reply to such traffic, so we want that the malicious traffic included in the trace to come from both the attacker and the victim.

3. **Non-determinism**: Multiple runs of the generator should not necessarly lead to the same trace, thus representing the non-deterministic nature of a network. By mostly consisting of machine-to-machine communication, the traffic in ICS is fairly periodic and stable, however, the quirks of the networks (failures, delay, etc.) are sources of non-determinism. For two generated traces, we want that the character-

istics of individual packets are not identical, while still respecting the overall per-flow empirical distributions observed in the original trace.

3.2.2 Related Work

Traffic generation is an important tool for analyzing the performance of networked systems [Cao+04; SB04; Kam+02; BDP10]. Typically, the generated traffic is used as workload, for example, to identify the maximum throughput of a particular network design. Many proposed solutions, such as [Cao+04], focus on uni-directional traffic generation in order to simplify the generation process, assuming that the traffic from a server to its clients dominate the perceived performance of the studied system and that therefore the other direction can be neglected. Such an approach is not realistic to evaluate IDS. In an ICS, both directions of the communication between a control server and a field device are equally important, therefore, in the evaluation of the IDS, both end-hosts must behave like they would in a real environment. In [Wei+06], the authors implement two-way communication by accurately replaying existing traces to assess the quality of a network objectively. In contrast, our goal is to deliberately generate traces that differ from each other in a certain way, so they can be used to evaluate IDS under different conditions. There exist several open-source tools that can be used to generate bi-directional traffic. Like [Wei+06], Tcpreplay [Kla30] replays network traffic previously captured and does not fulfill our need for non-determinism (Section 3.2.1). Iperf [30] is a speed test tool. While it could be used to replicate communication between two end hosts at flow-level, there is no way to directly control the Packet Size distribution (PS) and the Inter Packet Time (IPT) distribution, which is a desirable feature as we will discuss in Section 3.2.3. D-ITG [BDP12] allows to choose the PS/IPT distributions from a predefined set of distributions (Normal, Gamma, etc). In contrast, our approach is able to replicate arbitrary empirical distributions.

More related to traffic generation for IDS evaluation are [Spe+09; SYB04]. In [Spe+09], the authors model SSH brute-force attacks at a flow level with a Hidden Markov Model. However, they focus only on malicious traffic. In [SYB04], the authors present a framework to create attack profiles defined by exploits and the propagation method. When generating traffic, packet header and payload are built from blocks and a validator monitors the target responses to ensure their correctness. The work is extended in [SYB05] to integrate legitimate traffic into the generation but the communication is one-directional. Finally, there exist a few works on the generation of network traffic for the evaluation of IDS for ICS [Al-+14; Mor+11; LF16]. All those works can be classified as application-level generators, as they generate traffic such that the packet payload is correct according to the application protocol

specification. The protocol in question is Modbus/TCP [Org06]. In [Al-+14], the authors parse the list of detection rules of Snort to automatically generate Modbus/TCP packets with malicious payloads that match those rules. The authors of [Mor+11] and [LF16] generate synthetic Modbus/TCP traces with attacks mixed in. Although these approaches generate packets with correct application-level payload, they have to rely on virtual environments and testbeds to simulate the overall behavior of an ICS and, therefore, are not the most realistic.

Finally, there are some works on how to generate synthetic traces from another one [Rin+18; VL14]. They rely on statistical models and machine learning techniques to generate traces that have similar properties to the one they were generated from.

3.2.3 The Traffic Generator

3.2.3.1 Overview and Design Choices

As mentioned in Section 3.2.2, there exist many different traffic generator. They operate at different levels [BDP10]:

- *Application-level* generators emulate communication patterns of network applications. Detailed knowledge of the applications/protocols is required to create such a generator. In ICS such knowledge is available for a protocol like Modbus/TCP but many others are proprietary and undocumented.
- *Flow-level* generators replicate traffic characteristics on flow level, such as the number of packets and the number of bytes transferred per flow. Their advantage is that no knowledge of the concrete application protocols is needed.
- *Packet-level* generators replicate characteristics of individual packets, such as packet size (PS) and inter-packet time (IPT). Therefore, packet-level generators can be also used as flow-level generators but not vice versa.

Generators can also be differentiated by their generation process. It can either be model-driven, meaning that the generation is done according to a model built a priori, or trace-driven, i.e. the generated traffic is a replay of a previously recorded trace [Wei+06].

Our goal is to generate traces for the evaluation of Flow-based IDS. This would suggest that a flow-level generator is sufficient. However, there is no unique definition of a flow used by all researchers. Although many define a flow with the IP header five-tuple (source/destination addresses, source/destination port, protocol number), the IPFIX specification [CTA30] is much more

general. The traffic generator that we present in the following operates on packet level for maximum flexibility. Its input is a packet trace from which it creates a model of the packet characteristics for each connection found in the input trace. These models are then used to generate a new traffic trace.

At this point, we have to ask how closely the generated traces should resemble the original input trace. An ICS network can have many quirks that are unrelated to the actual control and automation processes, such as the set of TCP options used by the hosts or the way acknowledgment packets are generated. We do not want to create exact copies of the input trace; our goal is to create traces that can be considered as valid ICS traces, but are sufficiently different from the input trace so they can be used for IDS evaluation. For this reason, the models used in our trace generator only capture the PS and IPT distributions of those packets of the input traces that actually contain application data, meaning that signaling packets such as empty ACK packets are not taken into account. Of course, a correct traffic trace must also contain signaling packets; we emulate a virtual network in Mininet [LHM10a] to generate them.

Figure 3.7 gives the general workflow of our traffic generator. We explain the different steps of the generation process in the following.

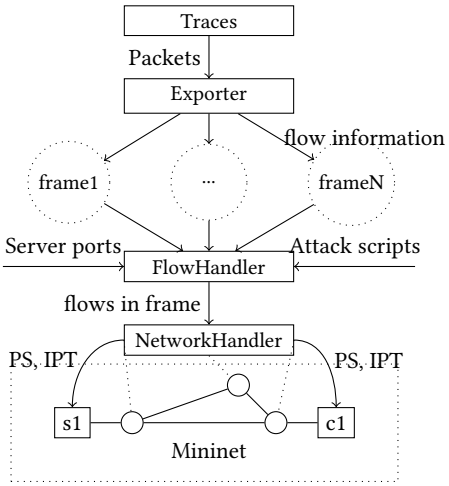


Figure 3.7: Workflow of the trace generator

3.2.3.2 Exporter

The *Exporter* reads an input packet trace and exports flow information that will be used by the next stage, the FlowHandler. The Exporter first splits the input packet trace into blocks of an identical duration, called *frames* in the following. This is necessary because the characteristics of the packets

may vary considerably from frame to frame: the behavior of a control and automation network system can depend strongly on the time of the day. For each frame, the Exporter performs the following steps:

1. Packets are grouped according to the traditional 5-tuple flow definition mentioned above, i.e., packets with identical source and destination address, source and destination port, and protocol number, are part of the same flow.

2. For each flow, general flow characteristics (number of packets, duration, etc.) are calculated and the list of PS and IPT values of all packets of the flow are built.

3. The list of flows and the computed flow data are forwarded to the FlowHandler.

Note that the Exporter has to be executed only once for an input trace. The flow information can be re-used whenever a new trace with similar characteristics should be generated.

3.2.3.3 FlowHandler

The *FlowHandler* processes the flow information exported by the previous stage frame by frame. Its first action is to group the flows into bidirectional flows, i.e. a flow is matched with the corresponding flow in the opposite direction. Then it determines for each bidirectional flow which host was the initiator of the communication. Like traditional Internet protocols, many ICS protocols follow a client-server model with the client being the initiator of the connection. Identifying who is the client and who is the server is not always obvious in a traffic trace if the protocol is UDP based or if the TCP handshake has been missed at the start of the capture, which is likely to occur for long connections (see Section 3.3). To cope with that, the user can provide a list of known server ports. If the FlowHandler finds a flow with ports that are not in the list, it considers that the source of the first packet in the bidirectional flow is the client host. Whenever the FlowHandler has identified a new flow in the frame, it instructs the next stage, the NetworkHandler, to create packets similar to those in the flow. For each flow, it will forward the following information to the NetworkHandler:

- The five-tuple identifying the flow. The IP addresses of the input trace are replaced according to a list of IP addresses provided by the user.

- The start time of the flow. The generator tries to replicate the start times of the flows from the original trace. In practice, this is not necessarily

done with perfect precision because the NetworkHandler needs time to configure the network emulation if a flow has not been seen before.

- The number of packets to generate for that flow in the current time frame. For each packet, the FlowHandler will sample the PS and IPT from the empirical PS and IPT distributions of the flow in the input trace.

- Whether the flow ends in the current frame. The FlowHandler peaks into the next frame to see whether the flow continues after the current frame. If not and the flow uses TCP, it will notify the NetworkHandler that the TCP connection should be closed after the last packet of the current frame.

To sample the PS, we estimate its probability mass function (pmf) from the histogram of the empirical data. Sampling new PS values from the pmf can be done efficiently since, as illustrated in Figure 3.4c and 3.4c, the communication in an ICS is often very regular with a few dominant packet sizes. We follow a different approach for the inter-packet times since time is continuous and even large input traces do not contain enough packets to obtain a useful probability density function (pdf) directly from the empirical inter-packet times: Instead, we use Kernel Density Estimation (KDE) to estimate the pdf.[1] KDE is a non-parametric technique that estimates the probability density function of a random variable by weighting the distances of all empirical data points for each possible value of the random variable [She04].

3.2.3.4 NetworkHandler

As explained above, the *NetworkHandler* receives instructions from the FlowHandler on how to generate traffic in the current time frame. It is important to note that these instructions are not enough to directly output a traffic trace: Details on the packet header fields, e.g. TCP sequence numbers and options, and information on signaling packets, such as TCP SYN and ACK packets, are not provided by the FlowHandler. In order to generate a complete and consistent network trace, the NetworkHandler maintains a virtual network emulated with Mininet [LHM10a]. Mininet is a network emulator leveraging Linux network namespaces to create virtual networks.

This approach has several advantages. First, instead of simply replaying the input trace and mixing it with attack traffic in a separate step, we generate both the normal traffic and the attack traffic (see the next sections) in the

[1] We could also fit a parametric distribution, e.g. a Γ distribution, but there is no guarantee that a distribution fitting one ICS would work well for another ICS. We decided to use KDE since it does not require prior knowledge of the shape of the distribution.

same virtual network. This is important if a host exchanging normal traffic with other hosts is also sending attack traffic or responding to attack traffic. Our approach guarantees that in both cases, all traffic sent by a host has the same TCP/IP stack fingerprint. Otherwise, normal traffic and attack traffic (as well as the response traffic) could differ in unwanted ways and influence the detection result, since the TCP/IP stack can have an impact on the characteristics of the packets (e.g TCP Options) and therefore influence the flow properties. Second, the usage of a network emulator simplifies the configuration and management of the virtual network. Mininet enables to configure the IP addresses of the simulated hosts without having to adapt the routing table of the machine since the emulated network is isolated from the machine hosting the generator. Being able to manipulate the IP addresses allows generating more realistic traces where communication occurs from hosts coming from different networks, for example, the control server contacting external servers for OS updates.

The NetworkHandler creates a virtual host in the emulated network for each IP address that appears in the instructions sent by the FlowHandler. This is done dynamically: When the NetworkHandler sees a new IP address, it runs a setup script that creates and configures a new host for that IP address. Once the host is ready, the NetworkHandler forwards it the traffic generation instructions that it has received from the FlowHandler for that source IP address. Since the hosts share a single filesystem and a single process ID space, the NetworkHandler can communicate with them through Inter-Process Communication (IPC).

The topology of the emulated network is relatively simple: We use a variation of the dumbbell topology [Kam+02] with three interconnected switches named *A*, *B*, and *C* in the following. New hosts are either attached to switch *A* or *B* depending on whether they have been identified as a client or a server. Both switches mirror their outgoing traffic to switch *C* where all traffic is captured and finally exported as the result of the generation process. Mininet uses virtual Ethernet pairs to emulate network links and we do not set any constraints(bandwidth, latency,..) on them for the accuracy of the IPT. In our experiments (see below), the virtual links are able to transfer data at a rate of at least 20GB/s.

3.2.3.5 Virtual Hosts

Once a virtual host is created and configured by the NetworkHandler, it awaits instructions on how and when to send traffic to other hosts. As explained in Section 3.2.3.3, instructions include information about the destination of the traffic, the start time of the first packet to send, the IPTs, and the PSs.

If the flow is a TCP flow, the host identified as the initiator of the com-

munication is responsible for opening the TCP connection to the destination, given that the connection has not been already opened in a previous time frame.

We want to emphasize that except for the first packet of a time frame, there is no synchronization between two hosts to decide whose turn it is to send a packet. Each host will send packets according to its list of PS and IPT values generated by the model. We motivate this decision by the fact that we are interested in evaluating Flow-based IDS, meaning that we aim to only replicate flow-level information from the input trace and not application-level behavior. A possible way to synchronize bidirectional packet exchanges using TCP sequence numbers was proposed in [BDP12]. However, it would require retaining much more information (such as the sequence numbers in both directions) from the original trace, while in our approach only the PS and IPT values need to be kept.

Since our traffic generator works on a packet level and not on an application level, the hosts fill packets with random content. However, TCP being a stream-oriented protocol, there is no guarantee that blocks of data sent through several `send` operations on the BSD socket will be sent as individual packets. Similarly, the receive operation at the receiver socket might treat the payload of multiple packets as one single big block. This would result in communication patterns that are very different from the input trace. To have strict control on the number and the size of exchanged packets, we disable Nagle's algorithm [Nag84] on all hosts. This algorithm is used in TCP to reduce the number of packets sent in the network by merging small packets into bigger ones. Furthermore, we include a four-byte value in each sent packet that indicates to the receiver how many bytes it should read at once from the stream.

3.2.4 Generating Attack Traffic

Since the goal is to evaluate IDS, the generated trace must include malicious traffic. In order to simulate attacks, the user has to provide a configuration file containing the start time of the attack, the IP address of the attacker and the target, and a script file executing the attack. When the FlowHandler has reached the time frame corresponding to the start time of the attack, it will notify the NetworkHandler which then creates the virtual host for the attacker (if it does not already exist) and instructs it to execute the attack script. The script contains the code that will perform the attack.

Any network attack that does not require information about application protocols can be performed as long as the script is provided. Typically traffic of attacks such as SYN Flooding, TCP Scan, or TCP hijacking can be generated. As mentioned earlier, since the hosts are emulated, they will react to

malicious traffic. For example, during a TCP Scan, the target victim sends RST packets if the destination ports of the attack SYN packet are not open.

Only generating network attacks is an obvious limitation of this approach (shared with all generators not operating on application level). Since we do not simulate any application software on the hosts, a packet containing, e.g., a buffer overflow attack would not have any impact on the emulation. Nevertheless, it can be detected if it is sufficiently different from legitimate traffic in terms of packet statistics, as shown in Section 3.3.2.2. Moreover, network attacks, such as scans or flooding, also occur in ICS, for example during the propagation of a worm.

3.3 Experimental Validation

Our experimental validation consists of two parts. First, in Section 3.3.1, we evaluate the characteristics of the traces generated by our generator without malicious traffic. We evaluate our trace generator on three input datasets. Second, we study the practical usefulness of our generator in Sections 3.3.2 through 3.3.2.5 for the evaluation of different IDS proposed in the literature. Some of them are specifically designed for ICS. In the original papers, they were evaluated on control systems communicating with emulated field devices.

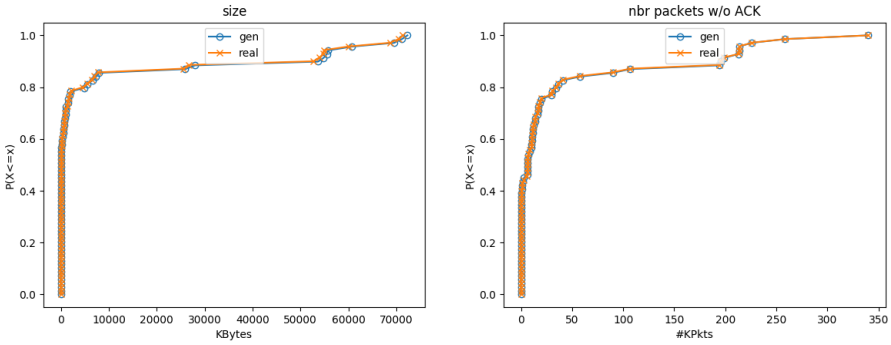
All experiments were run in the virtual machine provided by the Mininet project on a Dell XPS13 with 4GB of RAM.

3.3.1 Benign Trace Generation

3.3.1.1 Experiments with our BMS Trace

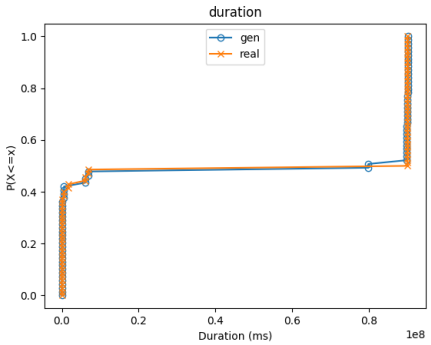
The first input trace is our BMS trace described in Section 3.1. We generate a one-day trace from it. The duration of a time frame is one hour. In the following, we focus on the communication between the control server and the gateways on port 2499 and the communication between the HMI and the control server on ports 50000 and 135, which are the HVAC-specific ports. This trace does not contain malicious packets.

We compare the flow-level characteristics of the input trace and the generated one. Figure 3.8a shows the flow-size distributions for both traces. As can be seen, the generator is able to mimic the distributions with an acceptable level of accuracy. It is important to mention that this comparison is computed only from the packet payload sizes. The headers are not taken into account. We can also see that the generated flows are slightly larger than the original flows. This is caused by the added four-byte marker (see Section 3.2.3.5). Figure 3.8b compares the number of packets observed per flow. Again, the numbers are close to the original values.



(a) CDF bytes per flow

(b) CDF packets per flow



(c) CDF of the flow duration

Figure 3.8: Comparison between generated and empirical flows

Figure 3.8c compares the duration of the flows. We observe that some flows take more time than their empirical counterparts while some take less time but the overall duration in the generated trace is close to the original.

Note that we have not taken TCP ACK packets into account in the above results since our goal is to obtain similar but not identical traces and not to replicate precisely every aspect of the original trace. We observe that fewer ACK packets are generated, though.

Then, we focus on the packet-level characteristics. We are particularly interested in a flow from one of the gateways to the control server because such flows have shown interesting variations over time in the original trace. Table 3.3 compares the mean and the standard deviation of the PS and IPT for this particular flow. The corresponding CDF are shown in Figure 3.9a and 3.9b. As can be seen, the error is small (again, note the shift by four bytes caused by the added four-byte marker). Since we have chosen a time

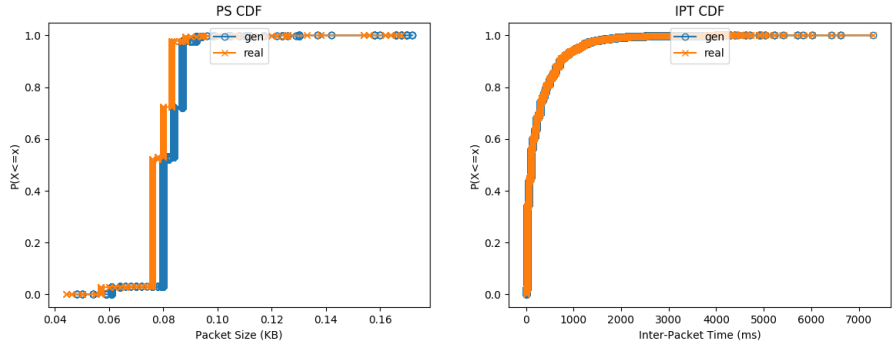
frame duration of one hour, the traffic generator recomputes the distribution of the PS and IPT every 60 minutes during the trace generation and so it is able to track the time dependency of the gateway's activity. This is visible in Figure 3.9c, which shows the number of bytes per hour sent by the gateway to the control server.

	Empirical	Generated
Mean PS (bytes)	78.352	82.359
Std PS	5.315	5.288
Mean IPT (ms)	264.169	265.03
Std IPT	397.846	398.789

Table 3.3: Comparison of the packet size (PS) and the inter-packet time (IPT)

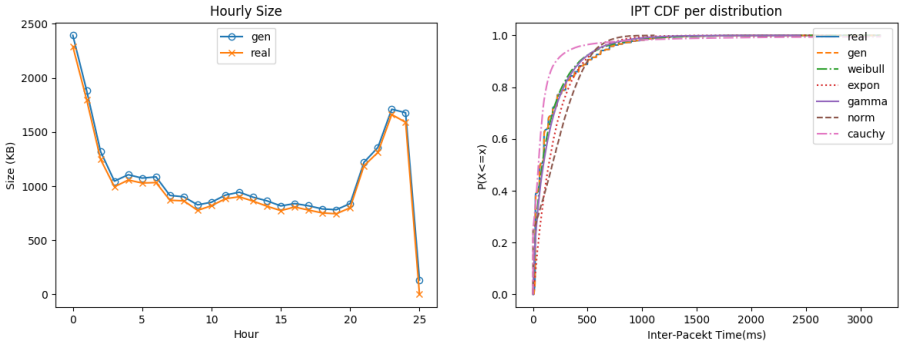
Then, we take a two-hour trace of the communication between the control server and the gateway and we use it to generate two hours of legitimate traffic with our generator. We compare the IPT of the real trace, our generated trace, and traces generated with D-ITG [BDP12]. Figure 3.9d shows the CDF of the IPT observed for our traces and the different distributions offered by D-ITG. Visibly, the trace generated by our generator (labeled "gen" in the plot) matches best the real trace. Another advantage of our generator is the fact that the generated traffic is bidirectional. Having bidirectional flows is more realistic with regard to how hosts communicate in ICS, which is necessary for a realistic evaluation of IDS (see Section 3.3.2.5).

Since our trace generator estimates PS and IPT distributions per (relatively large) time frames, it is not able to correctly reproduce short-range dependencies between packets. We have calculated the Auto-Correlation Coefficient (ACC) of the PS and IPT for the flows in the real trace and found ACC values relatively evenly distributed between -0.2 and +0.8 (PS; Figure 3.10) and between -0.8 and +0.8 (IPT; Figure 3.11). In contrast, since our generator draws samples randomly from the learned distributions, it produces uncorrelated series of PS and IPT, resulting in ACCs close to zero. This could be fixed by using models that correctly estimate short-range dependencies. However, we argue that this should not matter much for Flow-based IDS, which are the IDS we are primarily interested in. Flow-based IDS mostly base their detection on information collected from aggregated measurements. Furthermore, generating traces that behave similarly flow-wise but with differences in how the packets are exchanged can help to assess how sensitive an IDS is to small packet-level deviations.



(a) Packet size CDF of gateway

(b) Inter packet arrival CDF of gateway



(c) Bytes sent by the gateway

(d) CDF IPT of 2h trace w/ gen. and D-ITG distributions

Figure 3.9: Characteristics of the flow between a gateway and the control server

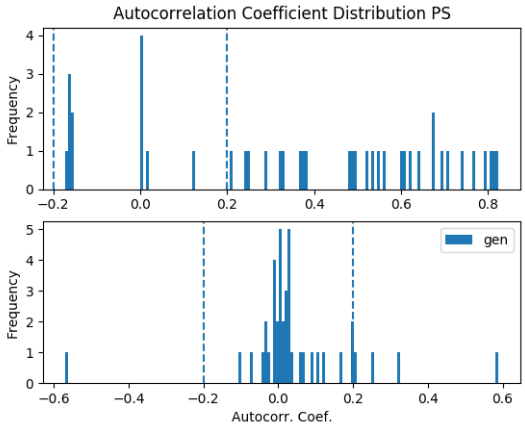


Figure 3.10: Distribution of the PS auto-correlation coefficient. Top: Real trace, Bottom: Generated trace.

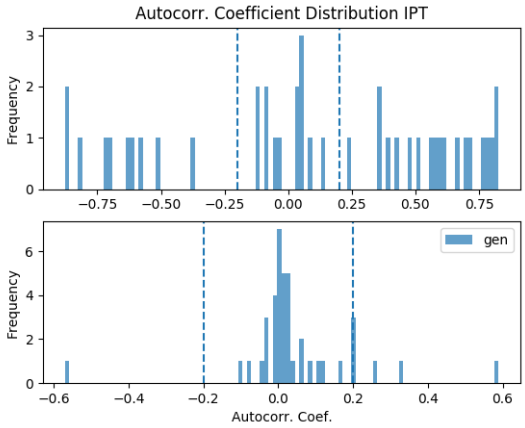


Figure 3.11: Distribution of the IPT Auto-correlation coefficient. Top: Real trace, Bottom: Generated trace.

3.3.1.2 Experiments with a Testbed Trace

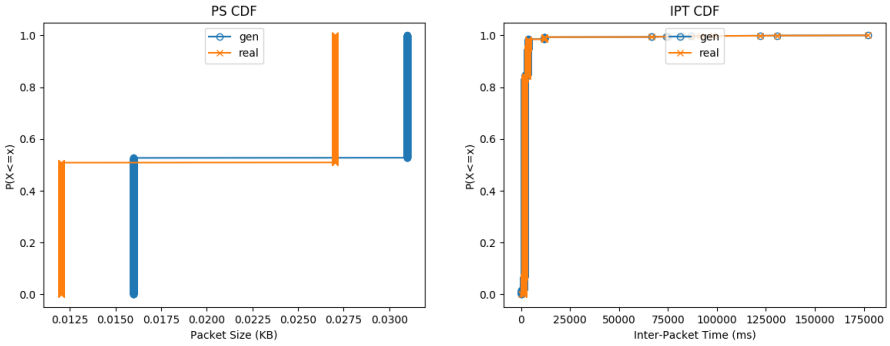
The next input trace has been generated from a public dataset collected at a testbed [Mor15]. The dataset contains Modbus/RTU messages exchanged between a Master Terminal Unit (MTU) and Remote Terminal Unit (RTU) in a gas pipeline control system testbed. The dataset is provided as a text file with the payload of the messages and their respective timestamp. We only keep legitimate messages and convert them to a packet trace by replaying the payloads in Modbus/TCP packets. For each packet in the obtained trace, we set its timestamp to the one in the original dataset. We generate a one-hour trace. Figures 3.12a and 3.12b show that the generated trace matches well the original trace.

3.3.1.3 Experiments with a Non-Public Real SCADA Trace

Finally, we have applied our generator to a SCADA trace from a utility company. The trace contains the traffic exchanged between MTUs and several dozen PLCs. Unfortunately, due to the confidential nature of the trace, we cannot provide further details. Again, we generate a one-hour trace. Figures 3.13a and 3.13b compare the PS and IPT CDFs of the generated trace with the original trace.

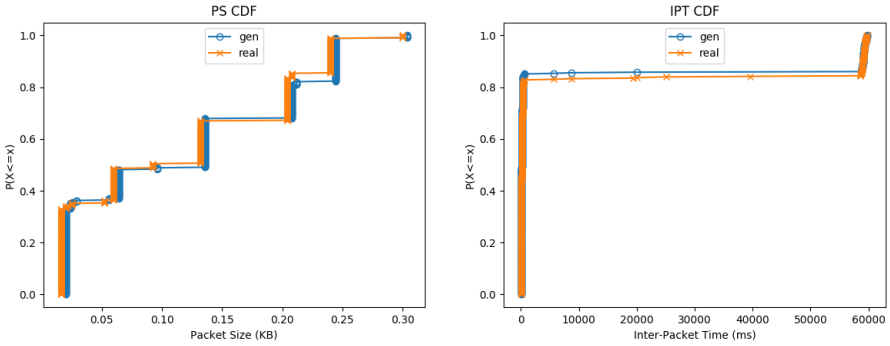
3.3.2 Generation of Benign and Malicious Traffic

After evaluating the realism of the generated traces, we want to measure their usability for evaluating Flow-based IDS. The goal of these experiments is to



(a) Packet Size CDF **(b) Inter packet arrival CDF**

Figure 3.12: Results for the testbed trace



(a) Packet size CDF **(b) Inter Packet arrival CDF**

Figure 3.13: Results for the non-public SCADA trace

show how the traces can be used to evaluate the detection performance of IDS and that the IDS do not exhibit unexpected behaviors when used with the generated traces, i.e. that the generated traces look "realistic" to them. To this end, we select several IDS from the literature, some of them explicitly designed for ICS environments, some not. Thus these experiment provide also some insight on the use of IT-IDS solutions in ICS. We focus more on having a diverse pool of IDS than having IDS with good detection performance.

We generate traces that include both legitimate and malicious traffic. The legitimate traffic is generated from our BMS trace and malicious traffic comes from two kinds of scenarios. The first scenario represents an attacker trying to obtain information about the ICS by performing horizontal and vertical scans. In this scenario, an attacker looks for hosts in a range of IP addresses

IDS	Scan H.	Scan V.	TCP Seq. Enum.
SCADA Pattern-based [VC09]	Yes	Yes	No
SCADA Flow-based [VC09]	No	No	Yes
EWMA-based [Hof+13]	Yes	Yes	No
Sketch-based [Mak+12]	Yes	Yes	Yes
Entropy-based [Liu+11]	No	Yes	Yes

Table 3.4: List of evaluated IDS and attack detection

(horizontal scan) and upon finding a host, the attacker scans it for open ports (vertical scan). The second scenario represents an attacker trying to inject a malicious packet into the communication between one of the gateways and the control server by spoofing the IP address of the control server and guessing the correct TCP sequence number [ikm30]. The attacker sends one packet per receiving window hoping to find the correct sequence number range. This kind of attack has more chance to work on older systems and with long connections, so it is safe to assume that it could happen in an ICS-like environment.

We have chosen these scenarios because of the different effects they cause on the network. The first scenario increases the number of flows in the ICS while the second scenario increases the number of packets in a flow. All the attacks were run against every IDS that are presented in this work but we only discuss the interesting results because some of the IDS can, by design, not detect all types of attacks. Table 3.4 lists the IDS and their detection capabilities with regard to the considered attacks.

3.3.2.1 Evaluation of a Pattern-based IDS

The first IDS proposed in [VC09] maintains a library of patterns for each host. A pattern is the set of IP addresses with which a host has communicated during a time window and the normalized number of packets exchanged with each IP. The library of a host is first empty and periodically updated every T seconds. After each period, a new pattern is created based on the network activities of the host in the past T seconds and that new pattern is compared with the existing patterns in the library. If a library pattern is similar to the new pattern, the former is updated. Otherwise, the new pattern is added to the library. The similarity test is controlled by a threshold value that we have set to 0.7 according to [VC09]. For each pattern, the IDS counts how often the pattern has been observed. The IDS raises an alarm if the tail probability of a pattern is lower than a predefined alert threshold θ.

We generate a two-hour trace where an attack is performed after around 45 minutes. The legitimate traffic is generated from our empirical BMS trace. The attack is a scan done both horizontally and vertically. The period size

is set to 180 seconds and $\theta = 0.5$. The attack starts at the end of the 13th detection period and runs until the end of the generation. The IDS detects the attack in the 15th period and all following ones. Figure 3.14 shows for the control server how many packets it has exchanged with each host (with host i being the attacker) before the attack (left) and during the attack (right). We repeat the trace generation and test the IDS also with a period size of 15 seconds, the attack starting after around 20 minutes, and other threshold values $\theta \in \{0.1, 0.2, 0.3, 0.4\}$. In all test runs, the IDS correctly identifies the attacking host and no false positive alarms are raised. This shows that our generator is able to produce traces that look "natural" to the tested IDS and that do not break any assumptions made by the IDS designers about the traffic.

As for the enumeration attack, the IDS is unable to detect it. Since gateways only communicate with the control server, the attacker bypasses the detection by spoofing the IP address of the control server when talking to the gateway.

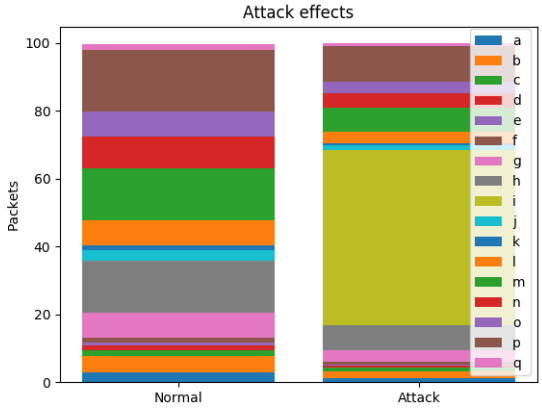


Figure 3.14: Number of packets exchanged by the control server with other hosts. Host i is the attacker performing a port scan of the control server.

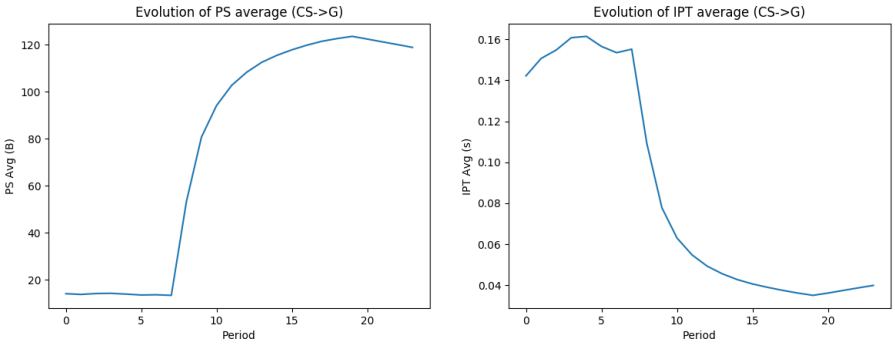
3.3.2.2 Evaluation of a Flow-based IDS

The second IDS proposed in [VC09] is a Flow-based IDS. The authors define a flow by the source IP address, destination IP address, and destination port. Notice that this flow definition does not consider the source port. The IDS computes flow information periodically and a database storing flow information over several periods is used to detect attacks. For each flow and observation period, the IDS computes the number of packets, the average, and the variance of PS and IPT. At the end of an observation period and when the packet count is large enough, the IDS runs a T-test for both PS and IPT.

The result of the test determines whether an alarm is raised.

The legitimate traffic is generated from our BMS trace. We use our generator to simulate the enumeration attack described in 3.3.2.

We set the period size of the detection process to 300s. The IDS needs to see flow information for a fixed number of periods to consolidate the record of historical flows before entering the detection phase. We set this number to 12. The attack starts in the middle of the 8th period and it is detected immediately when the IDS enters the detection phase. The attack impacts the PS and IPT statistics of the flow between the control server and the gateway. The PS average increases because the malicious packet (156 bytes) the attacker is trying to inject is bigger than packets observed under normal conditions. Correspondingly, the average IPT decreases due to the rate at which the attacker is sending the packet (approx. 1 packet every 50ms). The effect of the attack over time is shown in Figure 3.15. The reverse direction of the communication, i.e. the flow from the gateway to the control server is not affected. The attack is not successful so the gateway simply ignores the packets.



(a) Packet size average **(b) Inter-packet time average**

Figure 3.15: Impact of injected attack packets on the statistics of the flow from the control server to a gateway

While the Flow-based IDS is able to detect the attack in our test trace, it also raises false alarms. In total, the IDS generated 78 alarms for 52 flows whereof only 12 alarms were true positives. On average, 6.5 alarms were raised per period (without taking into account the learning periods). As explained, the IDS uses the T-test to determine if the average PS (resp. IPT) of a flow in the current period is similar to the historical average for that flow. The T-test compares a T-score $t_{calculated}$ computed from the flow statistics with a predefined standard value t_{value}. If the $t_{calculated}$ is greater than the t_{value} then the test has failed. An alarm is generated when at least one of the T-tests is rejected. Let F be a flow, $\text{avg}_F(\text{PS})$ the average of the packet sizes

of F in the current period, $\text{Havg}_F(\text{PS})$, the historical average packet sizes of F, $\text{var}_F(\text{PS})$, the variance of the packet sizes of F in the current period and pkts_F, the number of packets of F in the current period. The same values are defined for inter-packet time. $t_{calculated}$ is computed as follow:

$$t_{calculated} = \frac{\text{avg}_F(\text{X}) - \text{Havg}_F(\text{X})}{\sqrt{\frac{\text{var}_F(\text{X})}{\text{pkts}_F}}}$$

, where the X is either the packet sizes or the inter-packet times.

From what observed, the T-test is too strict for the generated trace. Even worse, when applied to our empirical, attack-free trace, the IDS raises 82 alerts. The reason is that the parameters of the T-test selected by the author are not suitable for our traces. The confidence level chosen is set to 99.9%. To reduce the number of false positives, we add a margin m such that an alert is raised when the T-score $t_{calculated}$ is greater than $t_{value} + m$. The margin m is computed in the following way: For each period, we take the average of the differences between the t_{value} and $t_{calculated}$ when there is an alert. The margin is then set to the maximum of those averages. Note that m is computed for both the PS and the IPT and for an attack-free trace. To evaluate the impact of m, we compare the number of alerts raised on the empirical trace (REAL), a generated attack-free trace (GEN), and a generated trace containing an attack (ATK). For each of the traces, we consider three ways to determine m for the PS and the IPT:

- m is set to 0 for PS and IPT;
- m is calculated for PS and IPT according to the method described above, using the REAL trace;
- m is calculated for PS and IPT according to the method described above, using the GEN trace.

Figure 3.16 shows the resulting number of alerts raised by the IDS when monitoring the three traces REAL, GEN, and ATK using the three ways to determinate m. As expected the number of alerts is the highest when m is set to 0. When the IDS is applied to the ATK trace, the number of alerts raised is exactly the same when m is either computed from GEN or from REAL, without a negative impact on the number of true positives (not shown here). However, this is not the case for the other traces: When the IDS analyzes the REAL and GEN traces, the number of (false) alarms is less when m is computed from GEN. In this experiment, we observed that the margin for the PS (resp. IPT) computed from REAL was 6.067 (resp. 6.075) while it was 8.58 (resp. 5.66) when computed from GEN.

Effectively, this means that our generated traces have helped us to find a better value for the parameter m of the detection process. This illustrates the usefulness of the generator; the generated traces allow testing IDS under slightly different but still realistic conditions and therefore can help to find more suitable parameter values. Having several traces is useful to find suitable parametrization of the test.

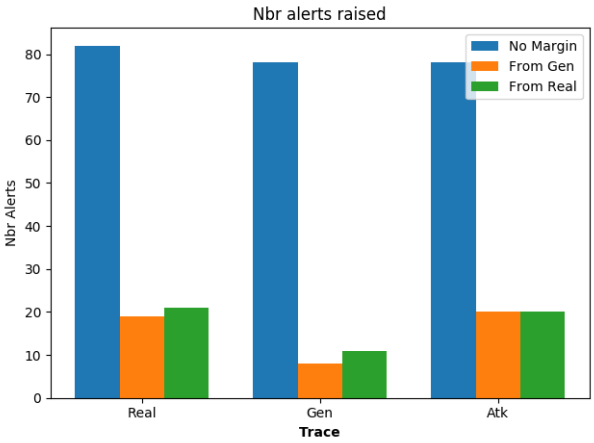


Figure 3.16: Number of alerts raised by the the Flow-based IDS on different traces without error margins resp. with error margins computed from two attack-free traces.

The IDS is not able to detect the scan attacks because the flows generated by them are too short-lived. The number of packets is too small to be able to perform the T-Test. In order to perform the T-test, the system needs historical values of a flow, the packet sizes and the inter-packet time. To get those historical values, at least one period is needed but the flows generated by a scan attacks do not last across multiple period, they are too short lived so the system can not perform the test. All the alarms generated were false positives.

3.3.2.3 Evaluation of a Flow-based IDS for Non-ICS Networks using Forecasting

The two IDS evaluated previously were designed for ICS-like traffic. We analyze in this and the following two sections three IDS that were not designed for ICS [Hof+13; Mak+12; Liu+11]. The two first IDS detect attacks by predicting the evolution of some flow metric (e.g number of flows seen) and comparing the prediction with real observations. The last one uses the entropy of the packet distribution to detect attacks. To evaluate these IDS, we use five traces: a two-hour long attack-free trace from the real BMS (labeled *trace1* in

the following) and four two-hour long generated traces containing horizontal and vertical scans (labeled *trace2* through *trace5*).

The first IDS we evaluate leverages the Exponential Weighted Moving Average forecasting technique (EWMA) to detect anomalies in flow metrics. We only consider the basic algorithm described in [Hof+13]. The IDS divides the time into intervals and predicts the next value

$$\tilde{x}_{t+1} = \alpha \cdot x_t + (1 - \alpha) \cdot \tilde{x}_{t-1}$$

of an observed metric from the current observation x_t and prediction $\tilde{x}_{t-1}$, where α is the weight of the moving average. Instead of directly comparing the forecasting error $e_t = x_t - \tilde{x}_t$ to a threshold, which could lead to many false positives, the authors calculate a cumulative sum

$$S_t = max(S_{t-1} + (x_t - T_{upper,t}), 0)$$

based on x_t and $T_{upper,t} = \tilde{x}_t + max(c_{threshold} \cdot \sigma_{e,t}, M_{min})$, where $c_{threshold}$ is a constant and $\sigma_{e,t}$ is the standard deviation of previous forecasting errors. When the sum exceeds the threshold

$$T_{cumsum,t} = c_{cumsum} \cdot \sigma_{e,t}$$

(where c_{cumsum} is a constant) an alarm is raised. In that case, the observed values are ignored so that the forecasting is not influenced by the abnormal values of the attack. In their evaluation, the authors look at the influence of the parameters $c_{threshold}$ and c_{cumsum} on the performance of the IDS. They show that c_{cumsum} has little impact, compared to $c_{threshold}$ which affects both the Detection Rate (DR) and the False Positive Rate (FPR). However, there is another parameter that can affect the IDS capability. The detection scheme of the IDS is to check if $S_t > T_{cumsum,t}$ in an interval t. During an attack, x_t increases and exceeds T_{upper} leading to an increase of S_t. However, when the attack finishes, S_t will still be higher than $T_{cumsum,t}$ for some intervals, meaning that for those intervals, alarms will be raised even though the attack is over. To avoid that, the authors mention an upper bound to S_t but they do not discuss its impact on the IDS. We call this upper bound S_{upper}.

We investigate the impact of S_{upper} on the Detection Rate and the FPR by modifying its value for different fixed M_{min} values. M_{min} is a margin used by the IDS to control the threshold in case $c_{threshold} \cdot \sigma_{e,t}$ is too small. The metric that the IDS forecasts is the number of new flows created per period. Note that a flow that lasts for several periods is not considered a new flow.

Figure 3.17 shows the Receiver Operating Characteristic of the IDS in experiments with $S_{upper} \in \{5, 15, 20, 50, 70\}$ for three different values of M_{min}. The other parameters are set to $c_{threshold} = 1.5$, $c_{cumsum} = 5$, and $\alpha = 0.01$. The interval size is set to 5 seconds. Several observations can be made. First,

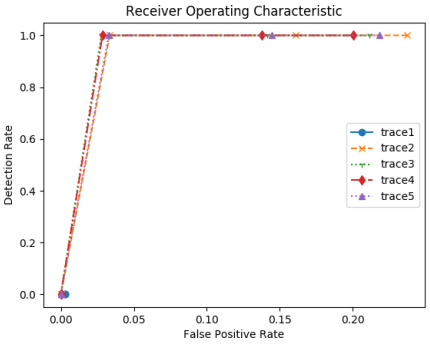
it is clear that S_{upper} affects greatly the FPR. A lower S_{upper} leads to a lower FPR. This is because S_t decreases faster after an attack which reduces the number of normal intervals for which S_t exceeds $T_{cumsum,t}$. However, if the upper bound is too low (e.g. 5 in our experiment), it might never exceed the threshold even during an attack meaning that attacks are never detected. This means there is a trade-off between DR and FPR. Second, when M_{min} is small (i.e. 0 or 5), the impact of S_{upper} on the FPR is not the same for the different traces. This is caused by the random nature of the generation process. Although the generated traces are identical in terms of high-level statistics, such as the number of flows, packets may end up in different intervals, which impacts the forecast and since M_{min} is low, it is the forecasting error that influences the most T_{upper} and therefore S_t.

Contrariwise, when M_{min} is large (= 20) the impact is more uniform among traces. This can be explained by the fact that when M_{min} is large, $T_{upper,t}$ tends to be large too, so S_t increases slower. Therefore, S_t does not necessarily reach S_{upper}. This also explains why the FPR is low when M_{min} is large since S_t exceeds T_{cumsum} less often. In the case where M_{min} is low, the IDS can reach a DR of 100% with an FPR of 20%. In the other case, it can detect 97% of the attack without raising any alarms. The results show that the value of M_{min} influences how sensitive the IDS is to the order in which the packets are sent. A Flow-based IDS uses aggregate metrics so one would expect that the order in which packets are exchanged should not matter much. However, thanks to the generated traces, we can see in Figure 3.17a that this is not always true.

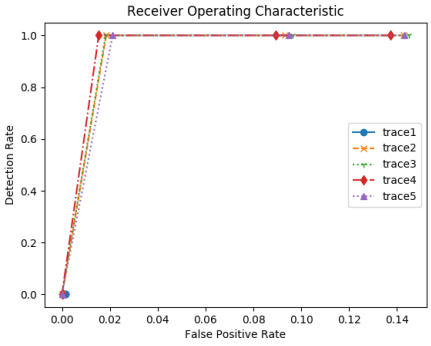
The IDS is not able to detect the enumeration attack since the attack does not impact the number of flows created.

3.3.2.4 Evaluation of a Sketch-based IDS

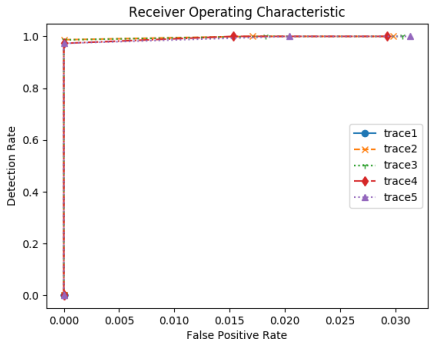
The IDS in [Mak+12] relies on a K-ary Sketch. This is an array of L mutual independent hash tables of size C. The table keys are IP addresses and the values are user-defined counters (e.g number of packets received). The time is divided into intervals of duration T and at the end of each interval ($k \cdot T$ with k a positive integer) the IDS does the following: For each counter, the IDS uses the Least Mean Square (LMS) to forecast the value of the counter in the next interval from the n previous values. LMS attempts to minimize the error between observation and forecast by using a gradient-based method of steepest descent. The predicted value is computed from a weight vector W, randomly initialized, and a vector of previous values. In each interval, LMS will correct W based on the error. At the end of an interval, for each hash table, the IDS computes two distributions: the forecast one and the observed one. The Chi-Square divergence is used to measure the distance between



(a) EWMA IDS, $M_{min} = 0$



(b) EWMA IDS, $M_{min} = 5$



(c) EWMA IDS, $M_{min} = 20$ (NB:x-axis scale)

Figure 3.17: IDS using EWMA forecasting with different M_{min} values

the two distributions. The IDS maintains for each hash table h two timeseries $\chi^2_{h,kT}$ and $\tilde{\chi}^2_{h,kT}$. $\chi^2_{h,kT}$ is the timeseries of the Chi-Square divergence computed up to the kT interval. $\tilde{\chi}^2_{h,kT}$ is a timeseries derived from $\chi^2_{h,kT}$ that does not contain the outliers from $\chi^2_{h,kT}$. To detect outliers, the IDS uses a threshold given by the Cheyseve inequality. A Chi-Square divergence d is considered an outlier if

$$d >= \mu_{h,(k-1)T} + \alpha \cdot \sigma_{h,(k-1)T},$$

where α is a constant and $\mu_{h,kT}$ and $\sigma_{h,kt}$ are the mean and the standard deviation of $\tilde{\chi}^2_{h,kT}$ respectively. They are updated dynamically with EWMA as

$$\begin{aligned} \mu_{h,kT} &= \beta\mu_{h,(k-1)T} + (1-\beta)\tilde{\chi}^2_{h,kT}, \\ \sigma^2_{h,kT} &= \beta\sigma^2_{h,(k-1)T} + (1-\beta)(\tilde{\chi}_{h,kT} - \mu_{i,kT})^2 \end{aligned}$$

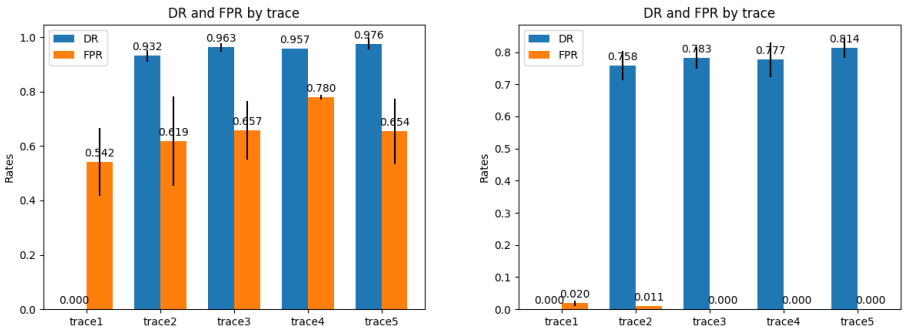
When an outlier has been detected, it will be replaced in $\tilde{\chi}^2_{h,kT}$ by the last value of $\tilde{\chi}^2_{h,(k-1)T}$. If outliers have been detected for more than H hash tables

and for η consecutive periods an alarm is raised. Observed values are ignored for the forecasting during an attack.

We evaluate the IDS with the traces described in Section 3.3.2.3. The results are showed in Figure 3.18a. The parameters are fixed: $L = 5$, $C = 100$, $n = 5$, $H = 3$, $\eta = 1$, α=4, $\beta = 0.7$. As mentioned before, *trace1* is the real, attack-free trace and the others are generated traces containing attacks. The weight vector of the LMS techniques is initialized randomly so we perform 10 runs of the experiment. The metrics computed is the number of packets received by a host.

The IDS is able to detect attacks with an acceptable rate (above 90%). However, the FPR is high in every case. This can be explained by the update method of the $\tilde{\chi}^2_{h,kT}$ timeseries. When an outlier is computed in a period, the last non-outlier Chi-Square divergence value is used as a replacement. This means that if an attack lasts for several intervals, the IDS will keep adding the same value to $\tilde{\chi}^2_{h,kT}$. This will decrease $\sigma_{h,kT}$ and therefore lead to a small threshold. A solution for this problem would be to use a lower bound for $\sigma_{h,kT}$. We apply this solution for the lower bound of 0.2 and show the results in Figure 3.18a. We can see that while this solution reduces the detection rate, it drastically decreases the number of false positives.

Apart from *trace1*, which is attack-free, the performance of the IDS is quite similar for every generated trace suggesting that, in contrast to the previous IDS, the order in which packets are sent has little impact on the performance of the IDS.



(a) Unmodified IDS **(b) IDS with lower bound**

Figure 3.18: Performance of the Sketch-based IDS

We evaluate the IDS with the second scenario and we observe similar performance. The IDS has a Detection Rate of 100% and a False Positive Rate of 17.4% when there is no lower bound for $\sigma_{h,kT}$. With the lower bound, the DR becomes 96.7% and the FPR becomes 0%.

3.3.2.5 Evaluation of an Entropy-based IDS

The last IDS evaluated relies on the PS distribution inside flows [Liu+11]. The motivation is that during a flooding attack, many small packets of the same size will be sent to the target host, leading to less diversity in the PS distribution. The authors define a flow as a 2-tuple (sip_i, dip_i) where sip_i is the source IP address and dip_i destination IP. Before running, the range of expected packet sizes must be defined. Then this range is manually divided into N bins of various size (e.g $[40, 60]$, $[60, 80]$, $[80, 120]$,...). Using these bins, the IDS computes an entropy score from the observed packet sizes:

$$H(dip) = -\sqrt{i_{max}} \cdot \sum_{i=1}^{N} p(x_i) log_2 p(x_i) (1 \leq i \leq N),$$

where $p(x_i)$ is the probability that the packet received by dip falls in the i^{th} bin and i_{max} is the bin with the highest probability. During an attack, the packet sizes will gather in lower packet size bins leading to a lower entropy score. The score is computed every T period and an alarm is raised if it is less than a threshold θ.

Applying the IDS directly to our traces results in an FPR of 100%. The reason for this is the fact that some hosts always receive packets of the same size, a typical behavior for control system networks. In order to make the IDS more suitable to our scenario, we ignore flows with an entropy score of 0 and assume that attack packets will differ sufficiently from regular packets to raise the entropy above 0 during attacks. We have determined empirically that the best results are obtained with many small bins for small packet sizes. For the experiments, we divided the range $[0, 20[$ into 10 bins of 2 bytes, the range $[20, 200[$ into 10 bins of 10 bytes, the range $[200, 1000[$ into 20 bins of 50 bytes, and finally the range $[1000, 1600]$ into 3 bins of 200 bytes. The period size was set to 60 seconds.

Figure 3.19 shows the Receiver Operation Characteristic for $\theta \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.8\}$. It shows that higher thresholds increase both DR and FPR for all traces. Again, this can be explained by the regular nature of communication in control systems. The higher θ, the more likely it is that an attack-free flows with packet sizes of low variability will be classified as attacks. This IDS is a good example of a solution designed for traditional ICT networks that is not suited for control and automation systems. The results show that even though the overall evolution of the rates looks similar for the different traces, there are significant differences in terms of performance depending on the trace. For example, with a threshold of 0.4, the IDS reaches a DR of 60% with *trace5* while it only detects 40% of the attack *trace2*. This shows that an IDS that is *per se* designed to only look at PS distributions can still be considerably sensitive to differences in packet timing and order.

Note that the results would be even worse if our trace generator was not generating bidirectional traffic flows. In a unidirectional generator, only one direction carries traffic while the other mostly consists of empty ACK packets of identical size, resulting in an entropy of zero.

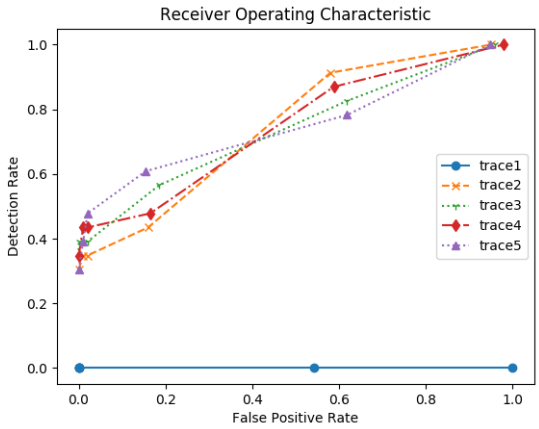


Figure 3.19: TrustGuard ROC

As far as the TCP sequence number enumeration attack, the IDS is not able to detect it when the threshold θ is below 0.4. In our scenario, the probes sent by the attacker have varying sizes meaning that the entropy does not decrease enough so that the IDS can detect the attack. With a threshold of 0.4, the IDS has a DR of approx. 18% and a FPR of approx. 11%. Similar to the result obtained with the scans attacks, a higher threshold leads to a better DR but also a higher FPR.

3.3.3 Practical Performance and Limitation

Our current implementation of the generator tool runs on a single machine. Given the characteristics of ICS, this is not a big issue. The low throughput allows estimating distributions on the fly and the presence of long-live flows reduces the overhead to open and close connections in the emulated network. Moreover, the stability of the ICS network topology causes that most of the virtual hosts are created at the beginning of the trace generation and, once they are created, changes to the emulated network are rare.

According to the classification in [BDP10], our generator can be seen as a hybrid flow/packet-level generator. It replicates PS and IPT distributions but it leaves the creation of the packets to Mininet. On one hand, this means that some information present in the original trace that could be useful for intrusion detection, like TCP/IP fingerprinting, is lost. On the other hand, this

allows users to experiment with other protocols (e.g. IPv6 instead of IPv4) by changing the configuration of Mininet. Since the scope of this work is the evaluation of Flow-based IDS, we have not discussed this point further.

3.4 Conclusion

Research conducted on ICS has shown that the characteristics of their network communication differentiate them from traditional IT networks to the point that researchers have designed ICS-specific Intrusion Detection Systems. Despite the large number of papers published on IDS for ICS, the state of the art in the evaluation of new detection algorithms is unsatisfactory: Nearly all existing publications rely directly or indirectly on experiments in testbeds or simulated environments [CDT21]. Those few that use real ICS have not published their datasets.

In this chapter, we have described and published a network trace collected in the Building Management System (BMS) of a university campus. The BMS depicts common characteristics of automation systems, such as long-lived connection, while also depicting diurnal patterns often associated with Internet traffic. To our knowledge, this is the first public trace of its kind. It can be downloaded from

https://github.com/hvacanon/HVAC-ANON

We have also presented a traffic generator framework and tool that can be used to generate traffic traces for the evaluation of Flow-based IDS. The tool takes as input a network trace and replicates its flow-level statistics as well as certain packet-level statistics. It supports the generation of benign as well as malicious traffic. We have provided experimental results to show the ability of the generator to correctly replicate various existing traces.

Finally, we have demonstrated the usefulness of the generator by evaluating five IDS found in the literature. Among others, our results show that the generator can help to understand and adjust the sensitivity of detection algorithms.

Flow-Whitelisting and Software-Defined Networking for ICS Security

4

In this chapter, we will discuss the usage of Software-Defined Networking (SDN) for the security of ICS. We will first show how SDN can be used to mitigate the drawbacks of *Flow-Whitelisting* and help its implementation in ICS networks. To this end, in Section 4.1, we fill first explain, based on work by Barbosa *et al.* [BSP13], what Flow-Whitelisting is and how it can be used for ICS security. Then we will give an introduction to SDN in Section 4.2. Finally, we will present our approach to dynamic Flow-Whitelisting using SDN in Section 4.3.

In Section 4.4, we will discuss another way how SDN can be used for ICS security: We will present and evaluate our method to mitigate eavesdropping attacks in ICS networks using SDN.

4.1 Flow-Whitelisting in ICS

As discussed in the previous chapter, Flow-based IDS are particularly interesting for ICS networks because they do not require understanding the application protocols used by each device and therefore do not suffer from the use of proprietary protocol which is frequent in control systems. Among the Flow-based IDS that exist, some researchers have considered following a Flow-Whitelisting approach [BSP13; Kan+14; Cho+15]. In this approach, an identifier is assigned to each communication flow. The ICS operator defines, either manually or automatically, the list of identifiers (the *whitelist*) representing the flows that are legitimate in the network.

A flow identifier is typically defined from the values of the header fields of a packet. In IP networks, a commonly used identifier is the IP five-tuple, i.e., the IP source address, the IP destination address, the source port, the destination port, and the transport protocol found in the header of the IP packet. When an IDS based on Flow-Whitelisting monitors a network, every

packet is associated with a flow identifier. If that identifier is not on the list of permitted identifiers, the packet is considered malicious. The main advantage of this approach is its simplicity. Its main limitation is that if an attacker is able to inject malicious packets with the identifier of a whitelisted flow, they will not be detected. Flow-based IDS that analyze other flow properties, such as the size of the flow, could detect such an attack. However, even with this limitation, Flow-Whitelisting is useful in many situations as it is able to detect frequently observed network attacks, such as network scans or illegal accesses from insecure devices to critical parts of an ICS.

Barbosa *et al.* [BSP13] propose and study the application of a Flow-Whitelisting approach in ICS systems. Because most of the network traffic is generated by automated processes and changes in the network topology occur rarely, Flow-Whitelisting seems like a viable approach. The authors base their study on two important characteristics of a whitelist, its size, and its stability. A large whitelist would be too hard to manage and an unstable whitelist would require frequent updates of the network adminstrator, which could lead to inconsistent states and many false alerts.

In their work, the application of the whitelist is split into two phases, the learning phase and the detection phase. During the learning phase, an initial whitelist is created automatically from a network trace of the system. There are two requirements regarding the trace: (i) it must contain exclusively legitimate flows, and (ii) a majority of the legitimate flows existing in the system must be part of the trace so that the initial whitelist learned from the trace is as complete as possible. Once the initial whitelist is built, it is used to identify malicious flows in the detection phase. By definition, malicious flows are those flows that are not part of the whitelist. When such a flow is identified, an alert is raised to warn the administrator who can decide what action to take. As mentioned earlier, the whitelist may not contain all legitimate flows as it depends on the quality of the trace used in the learning phase. Therefore, during the detection phase, a network administrator must manually update the whitelist to add legitimate flows that were missed during the learning phase and thus triggered false alerts.

The authors evaluate their approach on legitimate packet traces from three different ICS environments: two water treatment facilities and one electric and gas utility. At each SCADA environment, network packets have been collected for more than 10 days. During the analysis of the results, the authors show the viability of whitelisting concerning its size and stability. They show that a host tends to communicate with a small set of other hosts, which leads to a small whitelist. Typically, the number of hosts and the number of flows in the whitelist are in the same order of magnitude. As far as stability is concerned, they find that after one day the whitelist contains 60 % of the flows and there are no addition of flows up to the third day of collection. This

means that even if more than half of the flows can be discovered within a day, there is still a large part of them that shows up later which will cause false alerts. The authors assign each alarm to four classes:

Dynamic Port Allocation: This class consists of the alarms that are triggered because hosts changed the port number they were using in the past to communicate. The authors discover in the packet trace that there are times where several TCP connections are made by the same hosts in short sequence, with transport port numbers monotonicly increasing on both sides of the connection.

Manual Activities: This class includes the alarms caused by human activities. For example when a human creates a direct connection to a host to perform some manual operations.

New Host: These alarms are triggered when a new host joins the network.

Other: All alarms that are not part of the other classes.

How an alarm must be treated depends on the administrator who has to take a decision for every single one of them. This can be time consuming and not efficient. The authors study the possibility of Flow-Whitelisting but not its implementation in a real system. In [LRF16], the authors describe a practical implementation of a whitelist using the commercial IDS Snort. However, the whitelist is fixed once created and cannot be updated.

To overcome the above problems and limitations, we propose a practical implementation of whitelisting leveraging Software-Defined Networking in Section 4.3.

4.2 Software-Defined Networking

A network connects several hosts and allows them to communicate with each other. In order to do that, the network needs to perform two things. First, it must find a path between any pair of hosts that wish to interact, which implies that it should know how to reach a host from another. Second, it must carry and forward the messages that are sent such that they reach their destination. In computer networking, these tasks are referred to as the *control plane* and the *data plane*. The control plane includes all the decisions taken by the network to handle the traffic. Typically, it is the control plane that creates the routes in the network to know which path to use when an host wants to communicate with another. For example in an IP network, it is the control plane that builds and manages the IP routing table. On the other hand, the data plane includes the operations performed by the network to forward the packets sent by hosts to their destinations based on the decisions of the

control plane. In other words, the control plane is responsible for creating logical circuits in the network used by the data plane to carry data from a source to a destination.

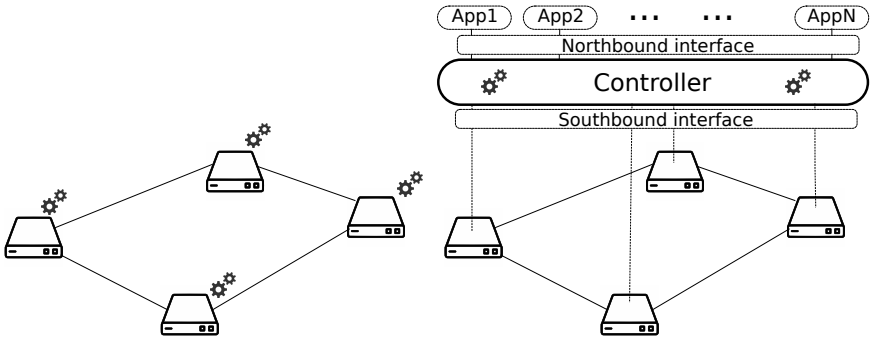
In traditional IP networks, IT or OT, both those planes are handled by the same networking devices which are the routers and the switches. Paths are created through distributed protocols (e.g., OSPF or BGP) that the networking devices use to exchange information so that they know how to forward packets. Having such a distributed approach increases the complexity in terms of configuration and management [Kre+15]. Indeed, in a modern network, it is not rare for operators to use network policies to specify the behavior of the entire network to reach specific goals, like performance or security, but since the control plane is part of the networking devices, each device must be configured individually. Moreover, it makes the network less flexible and it takes more time to deploy new technologies.

In recent years, a networking paradigm called Software-defined Networking (SDN) has emerged in order to fix those issues. In SDN, the control plane and the data plane are separated. The control plane is removed from these devices and transferred to a central controller. With this centralized approach, managing and configuring networks is easier since, at least from a management point of view, any change that must be done on the network is carried out only on the controller, not on the networking devices. In addition, the controller has a global view of the network and can perform changes dynamically.

4.2.1 Overview on SDN

As explained, the control plane and the data plane are separated in SDN (Figure 4.1). In this paradigm, the network devices act only as forwarding devices as they are only responsible for the data plane. The component of the network being in charge of the control plane is called the SDN controller. It implements all the control logic of the network and is logically centralized. Because the networking devices only manage the data plane, an interface is needed between both planes so that the decision taken by the control plane can be known by the data plane. Typically an Application Programming Interface (API) such as OpenFlow can fulfill this role. In SDN, such interface is referred to as the **Southbound Interface (SI)**.

Key features that are inherent to SDN are its flexibility and the programmability of the network. Indeed, the centralized controller has a global view of the network and can modify it through the southbound interface. The network operator can create applications interacting with the controller to get the state of the network and influence its routing decisions. The controller is seen as a "Network Operating System" used by network applica-



(a) Traditional network **(b) SDN network**

Figure 4.1: Network architecture

tions to dynamically manage the network, for example to enforce a particular forwarding policy. The communication between the network applications and the controller is done via what is called the **Northbound Interface (NI)**.

4.2.2 OpenFlow

The Southbound Interface (SI) defines the protocol between the controller and the forwarding devices as well as the set of instructions that forwarding devices can perform (e.g., forward packets, drop packets, ...). A well-known SI is OpenFlow [McK+08; BM14]. Openflow was initially proposed as a means for researchers to experiment with new protocols and encourage innovations. Later, it got standardized by the Open Networking Foundation. In order to use OpenFlow, three components are needed: OpenFlow-capable switches, a logically centralized OpenFlow controller, and a secure communication channel between the controller and the switches. OpenFlow has evolved since its first proposal and gained many new features. In the following, we only present the core concepts of OpenFlow.

In OpenFlow, the decision to forward a packet depends on the flow it belongs to and not only on its destination. It is common in SDN to take flow-based decisions for forwarding as it allows the same treatment for packets of the same flow regardless of the networking devices. In OpenFlow 1.0, each switch has a *Flow Table* where each entry is composed of three parts:

Header Fields: This part of the entry creates the mapping between the packets and the flow they belong to. It defines the set of fields used in each packet to identify the corresponding flow. The OpenFlow specification lists the different packet header fields that can be used for the mapping.

Action: This part of the entry specifies how the packet must be processed. Each packet that matches the entry header fields will be processed according to the specified action or list of actions. Common actions are forwarding packets to one or many switch ports, dropping packets, and sending packets to the controller. This last action is typically used in the case where a packet does not match any entry. The switch forwards it to the controller which can inspect the packet and later decide to add a new entry to the table.

Statistics: This part tracks several metrics related to an entry, like the number of packets that have matched the entry or the timestamp of the last match. These kind of statistics are useful to identify inactive flows and pinpoint the most active ones.

The OpenFlow controller populates and manages the table entries to enforce forwarding policies. The controller can be seen as a simple software that sends messages to the switches and receives messages from them according to the OpenFlow protocol. There are three classes of communication in OpenFlow [BM14]: controller-to-switch, asynchronous, and symmetric communication. The first class refers to all the messages sent by the controller to program, configure and manage the switches. They are also used by the controller to detect the features supported by a switch. The second class includes the messages sent by the switches to the controller without its solicitation. These types of messages are used by the switches to warn the controller that a specific event occurred like the arrival of a non-matching packet (see above) or an error. The last class consists of all the other messages sent from either side without solicitation. For example, either the switches or the controller can initiate echo messages to assess the availability of the secure channel.

OpenFlow, and SDN in general, has gained a lot of popularity because of its flexibility that allows to more easily deploy new network features and services. A recurrent concept that comes with SDN is Network Function Virtualization (NFV). A network function can be seen as a function provided by a node in the network infrastructure. Typical functions are firewalls, load-balancing , or IPS/IDS. Without NFV, such functions are usually handled by dedicated hardware placed at strategic locations in the network, like the edge, due to the lack of routing control. With NFV, these functions are virtualized, meaning that they are run as software, and because of the steering capability of SDN, they can be chained to offer new service in the network regardless of their placement [BM14].

So far, we have mainly discussed the advantages of SDN but there exist several drawbacks. First, like any centralized architecture, it introduces a single point of failure. The controller is the brain of the network and is responsible for any routing decision. If it is down the capabilities of the net-

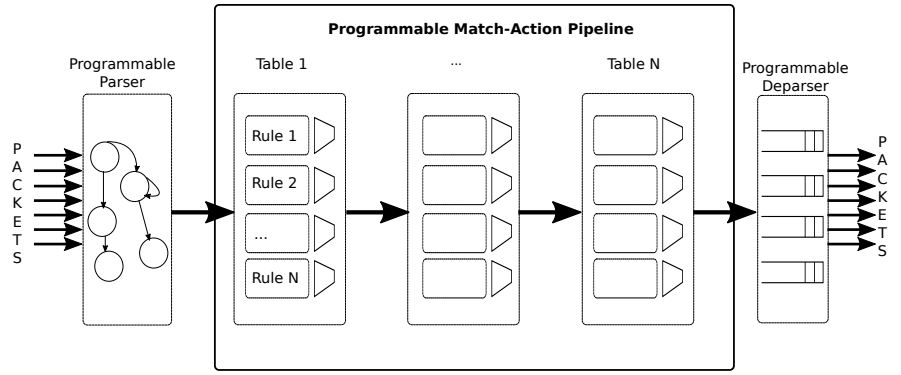


Figure 4.2: P4 Protocol-Independent Switch Architecture (PISA)

work are heavily impacted. Having several controllers is a way to decrease the risk that such an event happens but it increases the cost and complexity of the infrastructure. A second drawback, which is related to the first one, is the lack of independence of switches. When an unknown event occurs, like the arrival of a packet that they do not know how to process, they need to wait for the instructions of the controller or perform a default action, such as dropping the packet. In both cases, the host sending the packet experiences a delay.

A drawback, or limitation, specific to OpenFlow is the fixed set of header fields that can be used for matching, which does not allow to apply policies based on protocols that are not part of the OpenFlow specifications. In 2014, Bosshart *et al.* presented a solution to remove this limitation. They presented a high-level language for **Programming Protocol-Independent Packet Processors (P4)**. Their idea consists of having network devices that can be programmed to tell how to process incoming packets. Like OpenFlow switches, P4 switches use a table-like data structure and the southbound interface used to manage and populate them is called P4 Runtime [Ner18].

4.2.3 P4

Compared to OpenFlow which allows the programmability of the control plane, P4 goes one step further by allowing the programmability of the data plane as well. Specific hardware that understands the P4 programming language is required. The P4 language is based on an abstract model called the Protocol-Independent Switch Architecture (PISA) which is a high-level abstraction of a switch. P4 programs are implemented assuming that they will be compiled and run on switches following PISA, but others exist. PISA describes three components (Figure 4.2) which are the parsers, the table pipeline, and the deparser. Each of these components is programmable.

The parser determines which protocols are recognizable by the switch and therefore can be used for matching. The parser is implemented as a deterministic state machine where the states are the protocol headers and the transition determines their order of appearance in the packet. Depending on specific fields in a header, a transition goes from a state to another. For example depending on the Ethertype field of the Ethernet header, the parser knows if the next header is an IPv4 or an IPv6 header. Then by looking at the Protocol field of the IP header, it can detect if its a TCP or UDP packet and so on. A P4 program describes the headers as well as the transitions of the state machine, as illustratd in the below example program:

```
header ethernet_t{
        bit<48> dstAddr;
        bit<48> srcAddr;
        bit<16> etherType;
}
header ipv4_t{
        bit<4>  version;
        ....
        bit<32> srcAddr;
        bit<32> dstAddr;
}

struct headers {
        ethernet_t ethernet;
        ipv4_t ipv4;
}

parser MyParser(packet_in packet, headers hdr, ...) {
        state start {
                transition parse_ethernet;
        }

        state parse_ethernet {
                packet.extract(hdr.ethernet);
                transition select(hdr.ethernet.etherType) {
                        TYPE_IPV4: parse_ipv4;
                        default: accept;
                }
        }

        state parse_ipv4 {
                packet.extract(hdr.ipv4);
                transition.accept;
        }
}
```

In the first version of OpenFlow, there was only one flow table but latter versions allow sequences of flow tables against which packets can be matched. P4 uses a similar idea by defining a Match-Action pipeline. A P4 program de-

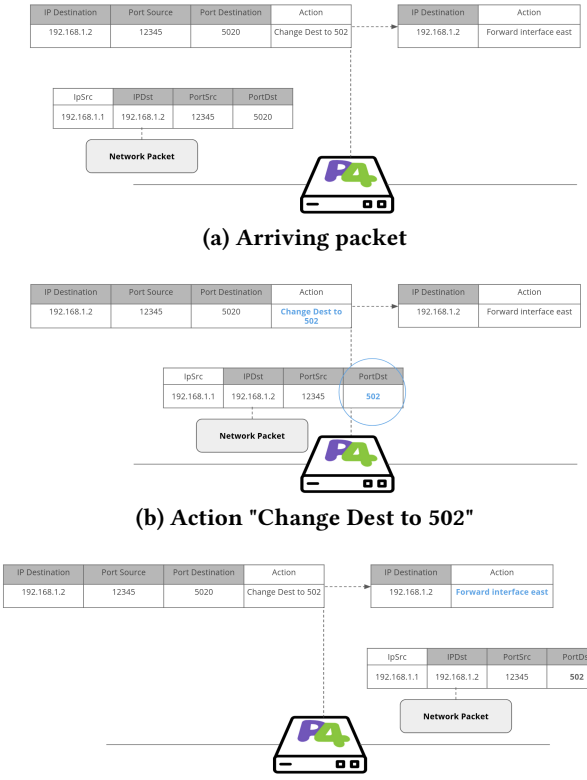


Figure 4.3: Example of P4 pipeline execution

scribes each table by selecting the headers to consider for the matching and the actions to perform in case of matching. P4 describes a set of protocol-independent primitive actions (add header, modify fields, drop, ...) that can be combined to create more complex actions. The P4 program specifies the flow of control between the tables in an if-else fashion where conditions are tested to perform different actions depending on the context. In addition to the the packet headers, P4 provides standard metadata that is related to the packet, e.g., the port on which it arrived, and that can be used as matching fields and/or in the control flow for the tables. Figure 4.3 shows an example of P4 usage to modify and forward Modbus/TCP packets. In this example, an arriving packet matching the destination port 5020 and the destination address 192.168.1.2 triggers an action to change the port and then a forwarding action.

After the parsing phase, packets are formatted according to an internal representation to the P4 switch that is easier to manipulate and to use with tables. At the end of the pipeline, packets must be turned back to a format that

can be sent on the wire. This is the role of the deparser which reassembles the packet by serializing its headers.

4.2.4 SDN for Security

The main motivation of SDN was to provide more flexibility and to ease the configuration of the network. These properties are tied to the routing and forwarding decisions taken by the control plane and therefore are also related to security. In a traditional IP network without SDN, a host can communicate with any other host at any given time as long as there is a path existing between the two hosts. In terms of security, this is far from ideal as there are no restrictions on which host can contact a given host. To change that, traditional IP networks use middleboxes such as firewalls or IDS to analyze and block traffic under certain conditions. Depending on the complexity of the networks, the number of hosts, or the topology, the placement of such middleboxes can be challenging. In order to be effective, these middleboxes need to be deployed such that every packet traversing the network must pass through them. Ensuring that results in high infrastructure and management costs [ILW06]. Moreover, the distributed nature of the control plane where the network devices take individual decisions, for example when there is a failure, can lead to unexpected situations where some packets are missed [She+12]. With SDN, such a situation can be avoided because of the programmability of the network and the centralized approach used for the control plane. The controller has a global view of the network which means that it knows how every packet in the network will be forwarded. Additionally, because it can dynamically modify the network forwarding policy, it can redirect traffic in any part of the network at any time, for example to send packets to an IDS middlebox before they are forwarded to their destination. With such level of flexibility, it becomes easier to ensure that a middlebox sees every packet in the network.

The usage of SDN to improve or implement security in computer networks has been studied in several publications. In the following, we will limit our discussion to work using SDN in IDS design and/or for ICS security.

OpenFlow-enabled switches can be used to collect Flow-based traffic statistics on the controller. Some researchers have proposed to use OpenFlow for Flow-based intrusion detection in a way similar to what has been done with NetFlow/IPFIX-enabled switches [So +10]. In contrast to the latter, the OpenFlow approach also allows implementing mitigation actions easily, such as whitelisting [Gio+14]. However, the flow data does not contain any payload or application-layer protocol headers and therefore limits the data analysis. Therefore, IDS based on OpenFlow monitoring works well for those attacks that have a distinct signature on flow level, such as DDoS attacks

[Lee+17]. [SG13] propose to extend OpenFlow such that the controller has full access to the packet payload. Then, they implement an application on the controller to detect port scans. However, they do not consider that sending full packets to the controller facilitates *control plane saturation attacks* [WXG15]. This attack is a type of Denial of Service attack which consists of generating a large amount of traffic such that it is redirected to the SDN controller to overload it. Such attacks can be easily performed through an SYN Flood attack as a lot of different flows are created during the attack. In the solution we present in the next section, the packets are inspected by an IDS and not by the controller, which prevents attacks from reaching the latter. This design is recommended by the OpenFlow authors [McK+08].

More specific to ICS security, [Don+15] suggest in a position paper to use SDN for the "dynamic monitoring" of smart grids, without providing further details. A very different application of SDN on ICS security is proposed by [Sil+15]: SDN is used to periodically reconfigure routes in ICS networks in order to make it harder for an attacker to eavesdrop or manipulate entire message exchanges. We discuss this further in Section 4.4.

Chavez *et al.* [CSP15] use the SDN controller to periodically change non-looping paths between endpoints. The issue with their solution is that paths are chosen randomly which can result in long paths and many rule installations on OpenFlow switches are required to change between the paths. Nayak *et al.* [NDR16] use SDN to schedule applications in ICS networks to provide them real-time guarantees. Their solution requires the involvement of hosts and is only suited for UDP applications.

4.3 Dynamic Whitelisting with Software-Defined Networking for ICS

We describe now our Two-Level Intrusion Detection System for ICS which relies on SDN and Flow-Whitelisting. The first level of the IDS is a whitelist-based filter for the Modbus/TCP protocol implemented in P4 on network switches. If a switch cannot find a matching whitelist entry for a packet, it forwards the packet to the second level of the IDS. This second level is a security engine running on a dedicated host. The engine determines whether the packet is malicious and instructs the controller to update filters on the switches to accept or block connections. In this way, most of the packet processing happens locally on the switches. The main contributions of our new design are:

- We implemented a packet processor in P4 for Modbus/TCP packets. Packet processing runs on the switches and thus eliminates the need of deploying additional IDS equipment (apart from the host running

the second-level security engine) or additional software on the PLCs or supervisory computers.

- By using a two-level design, we mitigate the major drawback of existing whitelisting approaches (see Section 4.1). Instead of directly rejecting suspicious packets caused by incomplete whitelists, they will be forwarded to the second level for further inspection.
- We show by experiments in an emulated environment the small impact of our design on communication latencies in the ICS, since most of the legitimate traffic will be handled by the switches without intervention by the second-level security engine.

Some two-level designs have been proposed by others [HGS18; MMS19], however they are not as flexible as our design since P4 allows to support new protocols without having to change the software on the switches. Furthermore, unlike other Whitelisting-based solutions, in our design a first traffic analysis is done on the switches and a second step happens in a central IDS. In this way, no dedicated monitoring components have to be deployed in the network and no additional software has to be installed on the PLCs or the supervisory hosts.

4.3.1 The Two-Level Intrusion Detection System

Our IDS operates on two levels. The first level consists of the P4 switches which are instructed to parse and match packets against whitelists. In order to reduce the possible number of false positives if the whitelists are not complete, non-matching packets will be forwarded to a dedicated host running the second level of the IDS. In this second level, the packet is analyzed by a security engine using Bro[1]. The security engine interacts with a SDN controller and can instruct it to add new entries to the whitelists on the switches in order to allow connections reckoned as harmless or to block packets from sources identified as malicious.

Our main motivation for using an SDN enabled network is that it provides easy management of whitelists and reduces the load on the IDS by performing most of the packet processing locally on the switches.

4.3.1.1 First Level: Whitelisting on the Switches

The first level of the IDS consists of matching incoming packets against a *flow whitelist* and a *Modbus/TCP whitelist* installed on the network switches. The whitelists are implemented as table lookups in P4. When a new TCP packet

[1] renamed to Zeek since 2018

```
table flow_id {
    reads {    // protocol fields defining a flow
        ipv4.srcAddr : exact;
        tcp.srcPort : exact;
        ipv4.protocol : exact;
        ipv4.dstAddr : exact;
        tcp.dstPort : exact;
    }
    actions {  // possible actions
        _no_op;
        add_miss_tag;
    }
}
```

Figure 4.4: P4 definition of the flow table

arrives, the switch first checks whether the packet's flow signature, consisting of both IP source and destination address, the TCP source and destination port, has a matching entry in the switch's *flow table*. The packet is immediately redirected to the second level if there is no such entry. Figure 4.4 shows the definition of the flow table in P4.

Then, the TCP ports of the packet are used to identify Modbus/TCP packets (default 502). The function code and the message length in the Modbus/TCP packets are matched against the *Modbus table*. This table contains a list of allowed Modbus/TCP function codes and the message length typically associated with them. Checking the code *and* the length makes it harder to perform a buffer overflow attack against a PLC. Packets with a matching table entry are forwarded to their destination, otherwise, they are sent to the second level. Because P4 switches are programmable, the whole process is highly flexible and can be extended to other protocols.

Prepopulating the Tables If the switches were started with empty tables, *every* packet would be forwarded to the second level, which then would (or not) instruct the controller to add table entries on the switches. However, our design also allows the controller to populate the tables beforehand with entries determined manually or by learning. In both cases, missing entries can be added later.

Based on their knowledge of their network, the ICS operators can specify which function codes (and message lengths) are allowed and should be preloaded into the Modbus tables. This way, Modbus/TCP messages using those function codes will not be sent to the second level. Similarly, the operator can add connection entries to the flow table to allow certain connections by default. In the basic design, a flow is defined by the IP source and destination address and the TCP source and destination port, therefore prepopulating

the flow table only works for TCP connections where both ports are known in advance, which is common for certain ICS equipment. If more flexibility is required, two flow tables have to be used, one for each direction: Entries in the first table do not specify the source port, and entries in the second table do not specify the destination port. In this way, the operator can prepopulate the tables with entries allowing all connections between an MTU and port 502 of a PLC, regardless of the source port used by the MTU.

To prepopulate the whitelists by learning, we follow the approach used by Barbosa *et al.* [BSP13] (see Section 4.1): During a learning phase, the traffic in the network is captured and used to build a list of allowed flows. Obtaining a fairly representative set of the traffic is much easier in an ICS network than in an IT network since the former is generally much more static in terms of used protocols and topology [R R12]. Nevertheless, the authors reported false positives because of flows occurring too rarely to be captured during a learning phase of reasonable[2] duration.

To automate the distribution of the whitelists, we give the controller the following information: (i) The globally unique ID of each switch, (ii) the IP addresses monitored by each switch, and (iii) additional information about the topology for routing.

For every packet in the captured training data, we extract its flow signature and tell the controller to add it to the flow whitelists of those switches that are expected to see the flow according to the routing information and the list of IP addresses. We also extract the function code and the size of Modbus/TCP messages. The size depends on the function and can be variable for some functions. For each non-error function code, we learn the message sizes observed in the training data. Thus, Modbus/TCP errors will always be reported to the second level. This is important because error messages can be an indication for the *Function Code Scan* attack whose goal is to discover the functions supported by a Modbus/TCP server. For each function–length pair, we also store the IP addresses of the hosts using them. Again, the controller will only upload Modbus table entries to switches in charge of handling the traffic from the respective IP addresses.

Figure 4.5 shows an example of an ICS network consisting of three subnetworks. We assume that the captured traffic contains message exchanges between a supervisory computer in the *Control Center* sub-network and PLCs in *Field Site 1* and *Field Site 2*, respectively. The controller will push the complete whitelist onto the switch of the *Control Center* and subsets of the list onto the switches of *Field Site 1* and *Field Site 2*. As a consequence, communication attempts between the two field sites will not be accepted by the switches.

[2]The underlying assumption of the learning approach is that the system is attack-free during the training phase, which becomes harder to guarantee if the learning phase is very long.

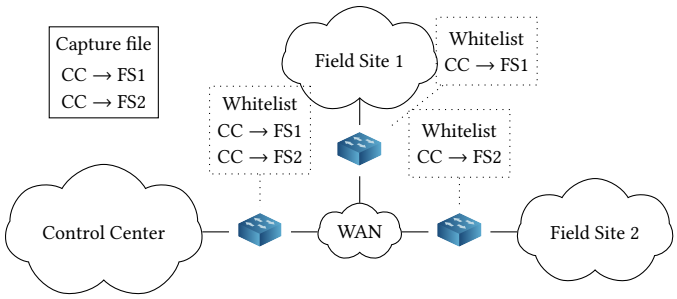


Figure 4.5: Whitelist distribution

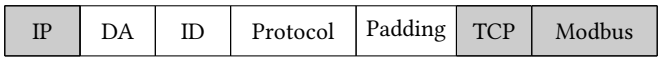


Figure 4.6: SRTag header

Packet Redirection to the Second Level To redirect a packet to the second level, the switches insert a new header between the IP and the TCP header, set the destination address to the IP address of the second-level host and change the protocol field in the IP header. The new header, to which we refer as the *SCADA Redirection Tag* (SRTag), contains four fields (see Figure 4.6):

- **Destination Address (DA; 4 bytes)**: the original destination address.
- **Identifier (ID; 2 bytes)**: the ID of the switch.
- **Protocol (1 byte)**: the original value of the protocol field.
- **Padding (1 byte)**

In addition, the packet length and the checksum in the IP header are recalculated.

4.3.1.2 Second Level: The Security Engine

The second level of the IDS consists of the security engine and *Bro* co-located on the same host. Bro is an event-based network security monitor. An event is triggered by a single packet matching a specific format or by a sequence of packets, such as a TCP three-way-handshake. Bro supports events for many protocols, including Modbus/TCP, and provides an interface allowing other applications to capture generated security events. [3] Our security engine communicates with Bro as well as with the controller. Since the security engine

[3] Other IDS, such as Snort, could be used; the only requirement is that the security engine can read their output.

is reachable by the switches it could be possible for an attacker to launch an attack against it, for example by flooding it with packets. In order to protect the controller from such a *control plane saturation attack*, the controller is located on a different host. In this way, the network operators can still control the network if the engine is under attack. The security engine performs the operations described below.

Packet Analysis The security engine analyzes packets sent by the switches in order to detect and repel attacks. Our solution favors passive monitoring and notification in form of alerts over active measures. In an ICS, a loss of communication due to a message wrongly blocked could be catastrophic. It is unrealistic to assume that the prepopulated whitelists are complete and cover all corner cases, such as Modbus/TCP messages only generated in emergencies. Therefore, our IDS will forward all packets that have successfully passed the analysis to their original destination, unless the analysis reveals that the packet is clearly malicious or malformed. Independently from this, the network administrator is always informed about newly appearing flows or function codes. When forwarding a packet, a special 64-bit tag is added to the packet to inform the switches along the path about the safety of the packet. The tag value is randomly chosen and made known to the switches by the controller. The following tests are performed by the security engine:

- Malformed Modbus/TCP messages where the packet size (without headers) does not match the message size, are dropped.
- Modbus/TCP messages requesting sensitive operations (such as a reset command) will be dropped if they do not originate from an authorized host. In practice, this is implemented by providing the engine the IP addresses of the MTUs. However, even if the message comes from an MTU, an alert is generated.
- Modbus/TCP messages whose source or destination addresses are not in the range of IP addresses of any field site or control center trigger an alert. Again, those address ranges have to be provided by the network operator.

In addition to forwarding the packet, a copy is sent to Bro for further inspection. The security engine listens for events raised by Bro, and depending on those it will take further actions that are described below.

Blocking Flows The security engine counts the number of malicious events received from Bro for each combination of source IP, destination IP, destination port, and transport protocol. This counter is also increased when

a Modbus/TCP error message is seen. We introduce a threshold called *max block request*. When the counter reaches this threshold, the security engine requests the controller to block further packets matching that particular signature. A blocked request is also issued when Bro sends an event that clearly indicates an attack, such as an SYN flooding attack. Note that the controller will only install a rule to block traffic on the switches if it does not overlap with an entry included in the original prepopulation list (see Section 4.3.1.1). Since we assume that all flows in the original list are legitimate, blocking them could seriously harm the operation of the plant controlled by the ICS. In any case, an alarm is raised to notify the operator about the situation. Flow blocking is implemented on the switches in form of a *blocking table* containing the source IP, destination IP/port, and transport protocol of the flows to block. This table is checked before the flow table and if there is a matching entry for the packet, it is dropped.

Adding New Entries to the Whitelists Provided that a non-error message has passed all security checks of the packet analysis, the engine can instruct the controller to add the function-length combination entry permanently to the Modbus/TCP whitelists on the affected switches. The underlying assumption is that this particular combination was missed when the whitelist was built. By adding an entry to the whitelist, the delays caused by the redirection to the second level are eliminated for further messages of the same type.

The addition of new entries in the flow whitelist is similar to the addition of new entries in the Modbus/TCP whitelist. Again, the underlying assumption is that this particular flow signature was missed when building the whitelist. Note that the controller will only modify the whitelists of the switches along the path relevant for the flow. Additions can be (a) disabled for maximum security, (b) done immediately, or (c) done only after a certain amount of unsuspicious packets have been observed for a flow. Option (a) is useful in situations where the ICS operator wants to closely monitor applications that are known to be particularly vulnerable. For example, some PLCs host HTTP servers for configuration and it could therefore make sense to never whitelist connections to port 80, even at the expense of the additional delay caused by the redirection. In our prototype, we whitelist a flow as soon as we observe a successful TCP three-way handshake, provided that the packets have successfully passed the packet analysis. SYN flooding attacks can therefore not fill the whitelists.

4.3.2 Evaluation

We now describe our evaluation of the two-level IDS. We first present the evaluation scenario and the test setup, followed by the results obtained for different test cases.

4.3.2.1 Scenario and Adversary Model

In our evaluation, we emulate a scenario where a malicious host under the control of an attacker is present in an ICS network. The attacker targets the PLCs. As they do not authenticate the commands they receive, they represent convenient primary targets for attacks.

The emulated ICS is shown in Figure 4.7. It consists of an MTU and six PLCs located in two field sites. The security engine and the controller are located in the north. Furthermore, the network contains a Human-Machine Interface (HMI) and the host controlled by the attacker. All hosts are connected through P4 switches to a core network. The switches are represented by boxes in the figure. The core network (the "diamond" in the center) consists of four routers, represented as cylinders. Typical for large ICS installations, the core network has redundant links in order to improve reliability.

4.3.2.2 Setup

To evaluate the performance of our IDS in the above scenario, we use the network emulator Mininet [LHM10b]. It emulates P4 switches using *bmv2*, a P4 implementation in C++. We use the Modbus/TCP library *pymodus (v1.3.2)* to emulate Modbus/TCP servers (the PLCs) and a Modbus/TCP client (the MTU). Each link in the network has a 10Mbps bandwidth and a delay of 5ms. Figure 4.7 shows the topology; the dashed lines represent the control-plane communication between the SDN-Controller and the P4 switches. We use Bro version 2.5.2. All the experiments were performed on the virtual machine provided by Mininet. The VM runs on a Dell XPS 13 with Ubuntu 16.04. The MTU sends requests to all PLCs every two seconds in order to obtain status information or to update process parameters. Four function codes are used in the communication between the MTU and each PLC.

4.3.2.3 Whitelist Management

Our first test shows the (undesired) effect of incomplete whitelists. We first capture a specific number of packets from the network (without any IDS or attacks running) and use the captured traffic to build the whitelists for the switches. Obviously, the shorter the taken sample, the less complete the whitelists. In a basic whitelist approach with static whitelists, an incomplete

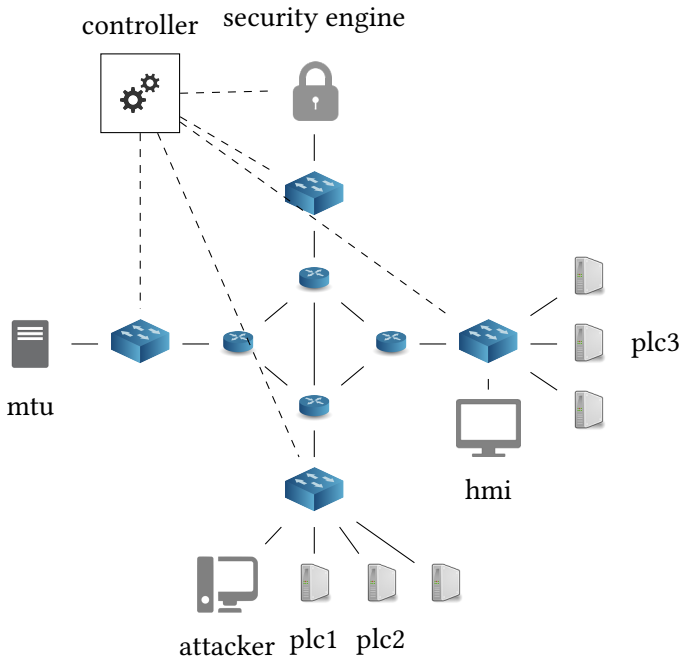


Figure 4.7: Setup topology

whitelist will result in a large number of false positives: every packet without an entry in the list will be blocked. In our two-level IDS, only the first packet of a flow or Modbus/TCP message type (i.e. a combination of Modbus/TCP function code and message length) will raise an alert. After the first alert, our IDS will add a new entry to the whitelist, in this way ensuring the functioning of the ICS.

Figure 4.8 shows the resulting percentage of false positives obtained for the basic whitelist approach and our IDS when using different capture file sizes. In total 937 packets were monitored.

4.3.2.4 Delay Measurements

In our second experiment, we measure the round-trip times (RTT) of request/response pairs between the MTU and PLC1 resp. PLC2. Note that we define RTT as the total time seen by the MTU between sending the request and receiving the response from the PLC. For the communication with PLC1, we have prepopulated the tables with matching flow entries, but without matching Modbus entries for two function codes used by the MTU. For PLC2, we do not preload any matching entries onto the flow tables nor onto the Modbus tables. The IDS is configured such that it will install new flow entries or Modbus entries. The measured RTT over two minutes is shown

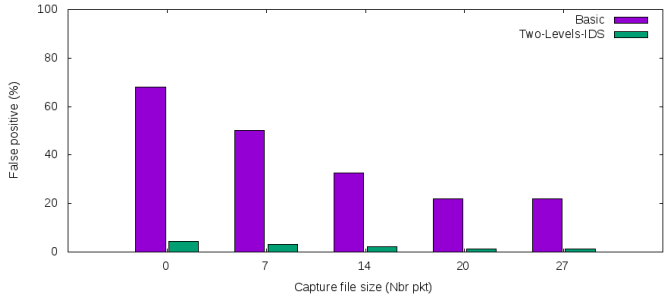


Figure 4.8: Number of false positive for different capture sizes

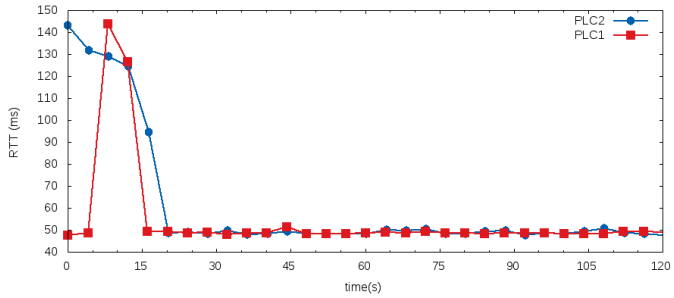


Figure 4.9: Round-trip time experienced by two flows. The flow to PLC1 has no matching Modbus entry at the beginning. The flow to PLC2 has no matching flow entry nor matching Modbus entry.

in Figure 4.9. The high peak in the RTTs for PLC1 reflects the fact that the message exchanges result in Modbus/TCP packets redirected to the security engine. At around time $T = 10$ seconds, the IDS has learned all combinations of function codes and message lengths used between the MTU and PLC1 and added them to the whitelists. After that, the RTT drops to its normal value. It should be noted that both the request packet as well as the response packet are affected: although they contain the same function code, they differ in length. We are aware that a real-time control system would suffer from the redirection, however we expect that a high percentage of the flows will be prepopulated so the number of flow redirected is kept to a minimum. Also, we think it is better to handle unexpected flows with delay instead of completely blocking them. For PLC2, the effect is even more pronounced because it does have neither a matching flow entry nor a matching Modbus entry. Therefore, the handshaking packets of the TCP connection are also affected by the redirection until the IDS gets a validation from Bro to add a new entry to the flow table.

Next, we instruct the security engine to never add whitelist entries for PLC2 to simulate a "sensitive" service that the network operator wants the

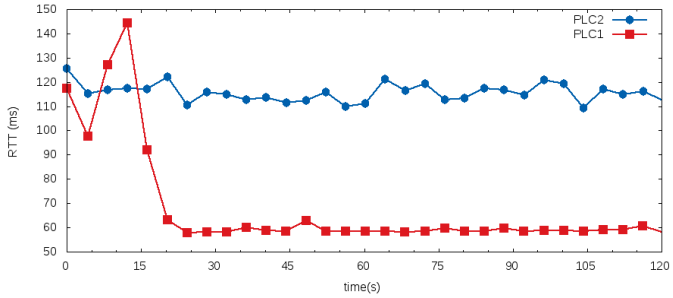


Figure 4.10: Round-trip time experienced by two flows. The flow to PLC2 is never added to the whitelist.

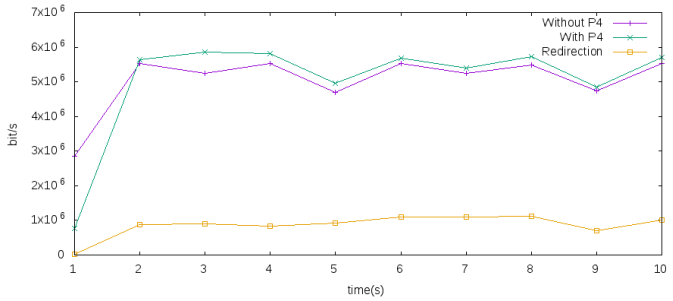


Figure 4.11: Impact of the switches on the throughput

IDS to continuously monitor. Figure 4.10 shows the different RTT experienced when communicating with the two PLCs. Unlike the previous experiment, no flow entry is added for PLC2 and therefore every packet is redirected to the second level.

4.3.2.5 Throughput Measurements

We measure throughput by sending data from the MTU to the HMI in three configurations : (1) both are connected via a 10Mbps link with a delay of 30ms, (2) the flow is added to the whitelist after the TCP handshake (3) each packet is redirected to the IDS.

We use *iperf (v2.0.5)* to generate traffic for 10 seconds from the MTU to the HMI. Figure 4.11 shows the achieved throughput. The results for configurations 1 and 2 are quite similar, except at the beginning due to the redirection of the three-way handshake. In configuration 3, the throughput is clearly affected by the redirection and stays at around 1Mbps. This is the cost of not adding a flow entry to the whitelist. Note that only the flow whitelist is used in this experiment because the traffic is not Modbus/TCP.

Table 4.1: Function Code Discovery

Max Block Request	# of Function Codes Discovered
1	7
3	9
5	11
∞	19

4.3.2.6 Blocking Scan Attacks

To test the protecting capabilities of the IDS against attacks, we study a scenario where an attacker was able to infiltrate one of the field sites and performs attacks. In Modbus/TCP, when a server is not able to perform the action requested by the client, it will send a function error code equal to the one in the request incremented by 128 along with an exception code set to 01 to specify the nature of the error. During a *Function Code Scan* attack, an attacker sends Modbus/TCP requests to a server (i.e. a PLC) to discover the function codes supported by the server. Depending on whether or not it receives this exception code, the attacker knows if the function code is supported by the server. As stated earlier, function error codes are never inserted in the Modbus/TCP whitelist so these events are captured by the IDS. We set the *max block request* threshold that is used to block flows (see Section 4.3.1.2) to 1, 3, 5 and ∞, respectively. Table 4.1 shows the number of function codes that an attacker could discover. The *pymodbus* server that we use only supports the 19 most common function codes [Org06]. Since there are gaps between those codes, an attacker can only discover 9 of them before it is blocked when the threshold is set to 3. We have used the *Function Code Scan* as an example to demonstrate the blocking capability of the IDS. Other Modbus/TCP scanning attacks, like a scan of the register addresses supported by a PLC, can be blocked, too, with this mechanism.

4.3.2.7 Blocking Sensitive Function Codes

We test how the IDS reacts to the use of sensitive function codes by an attacker. In this experiment, the attacker sends a Modbus/TCP request to a PLC with a sensitive function code. The function code is the *Force Listen Mode* function. When receiving such a request, a Modbus/TCP server will enter in the so-called *Listen Mode*, meaning that it will stop sending responses when receiving a request from any client. Figure 4.12 shows the number of packets exchanged between the MTU and a PLC per second. In the upper plot, the attacker successfully performs the attack at time $T = 30s$. At that point, the MTU stops receiving responses from the PLC. After a few attempts, it closes

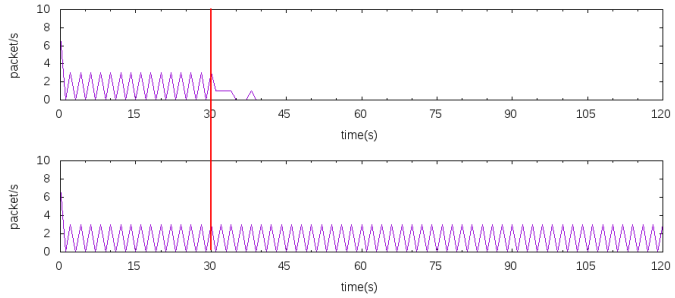


Figure 4.12: Number of packets transmitted during a *Force Listen Mode* attack against a PLC. In the above plot, the attack is successful and the PLC stops communicating.

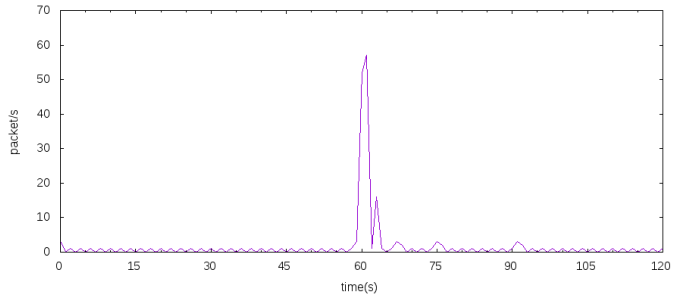


Figure 4.13: Number of packets received by a PLC during a SYN Flooding attack

the TCP connection. In the lower plot, the IDS detects the use of the sensitive function code by the attacker and blocks the flow. The MTU can continue communicating with the PLC.

4.3.2.8 Blocking SYN Flooding Attacks

In our last experiment, the attacker performs an SYN flooding attack against a PLC. To this end, it sends 60 SYN packets/s during 10 seconds but does not respond to the SYN+ACK packet. Even at a slow rate, the simple fact of sending more packet than usual to PLCs has important impact on their performance[Nie+18] Figure 4.13 shows the number of packets received by the PLC. The attack is clearly visible over the baseline of requests sent by the MTU every two seconds.

Since those flows are incomplete and not in the whitelist, the SYN packets will be redirected to the security engine. The redirection of those packets allows Bro to detect the attack and raise an event that will be received by the security engine. The latter will then issue the controller to block requests.

For this experiment, Bro was configured to raise an event if it sees more than 30 incomplete connection attempts per 5 seconds. The flow tables are not impacted by this attack since incomplete TCP handshakes do not cause installations of new flow entries.

4.3.3 Backup Flow Management

The efficiency of the original whitelisting approach is highly impacted by the quality of the learning trace which must include as much legitimate flows as possible. However, it is unlikely that this will contain all possible legitimate flows in the ICS because some of them do not occur under normal conditions. Examples of such flows are backup flows. To improve resilience, ICS typically uses backup PLCs so that when one of the main PLC crashes, the backup PLC takes over [BSP13]. When such an event occurs, the flow existing between the MTU and the main PLC disappears and the system switches to a new connection between the MTU and the backup PLC.

Since backup flows are new flows, they will get the same treatment as malicious one, i.e., they will be redirected to the second level and be delayed. This is undesirable as these flows are used by the MTU to get the state of the physical process. To avoid such treatment, these flows must be identified as backup flows and directly be added to the whitelist. We use the IP 5-tuple to define a flow of which only four pieces of information are known in advance for a backup flow: the IP address of the MTU, the IP address of the backup PLC, the port of the PLC and the transport protocol. As the client (the MTU) chooses its port randomly when connecting to the PLC, it is unknown at first. The reason we use the 5-tuple is to be more strict on the allowed communication between each host. However, to allow a faster establishment of the new flow, the flow definition is relaxed when the second level of our IDS thinks that a backup flow will likely occur.

4.3.3.1 Backup Flow Detection

Backup flows result from the termination of main flows. This means that by detecting such event, the second level can expect that a backup flow will be created.

In TCP, as used by Modbus/TCP, a straightforward way to detect the end of a connection is the RST (Reset) TCP flag and the FIN flag used in the four-way handshake to gracefully terminate a connection[4]. The two-level IDS will monitor the occurrence of such events, however there may be cases where

[4]For ICS protocols based on the connectionless UDP protocol, a different approach would be needed, for example based on timeouts. However, in DNP3, a popular ICS protocol based on UDP, PLCs can send messages at any time without necessarily following a timing rule. We leave the handling of such protocols to future work.

the crashing PLC may not be able to close the connection according to the TCP specification. In those situations, the PLC stops responding to the MTU which will lead the MTU to retransmit its data until they get acknowledged or until they are retransmitted too many times[5]. Considering all that, there are three events that the IDS needs to look for and in order to do that, we will use the P4 switches. First, every packet which belongs to a flow included in the whitelist and contains the RST or FIN flag will be cloned and sent to the second level. That way, the second level is notified that the flow is over in at least one direction. For the detection of retransmission, we use the capability of P4 switches to store metadata. For each flow, the metadata will store the sequence number of the last packet of the flow carrying data. This sequence number is used by the switch to identify retransmitted packets in the following way: the sequence number of the last packet carrying data (*lastSeqNumber*) is compared with the sequence number of the current TCP packet (*pkt.SeqNumber*). If they are identical, the switch considers that a retransmission has occurred and will send a cloned version of the packet to the second level. A TCP host can send multiple packets without receiving an acknowledgment in between, therefore, in case of retransmission, there may be the situation where *lastSeqNumber* is higher than *pkt.SeqNumber*. The switch also checks this case while taking into account that no wrapping around of the sequence number has occurred. The pseudo-code for the detection of the retransmission is shown below. Note that *lastPktLength* and *MAX_SEQ_NUMBER* are the length of the TCP payload of the last packet carrying data and *MAX_SEQ_NUMBER* is the maximum possible sequence number(2^{32-1}). Those values are used to detect the wrap-around case. For each packet, the appropriate values will be updated if the packet carries data.

```
if ((lastSeqNumber == pkt.SeqNumber and carry_data(pkt)) or
(lastSeqNumber > pkt.SeqNumber and
lastSeqNumber + lastPktLength > MAX_SEQ_NUMBER))
        clone_and_send;
if (carry_data(pkt))
        update(lastSeqNumber, pkt.SeqNumber);
        update(lastPktLength, pkt.length);
```

It is important to mention that this process is only performed by the switch which is closer to the source of the packet to avoid that all the switches along the path clone the packet to the IDS.

Upon receiving a retransmitted packet, the second level determines if it should consider that the flow has abruptly been stopped or not. To do so, the second level keeps a counter of retransmissions for each main flow. When this counter reaches a threshold t, the second level will start the backup flow

[5]The maximum number of packet retransmissions is OS dependent.

initialization. To avoid that the counter reaches t because of general network unreliabilities (e.g., packet losses), the counter is periodically reset.

4.3.3.2 Initialization of Backup Flow

As mentioned previously, the backup flow should be directly added to the whitelist. However, normally, the required information to create a 5-tuple flow entry is known only when the new flow is created. Unfortunately, we cannot wait that long, otherwise the connection establishment of the backup flow would be impacted by the redirection, which is something that we want to avoid as this flow is meant to allow the system to recover from a failure.

We therefore add two new table to the switches which are called the *backupTable* and *backupTableRev*. The rules in these tables match against the same IP 5-tuple except for the client source port. When the second level detects that a main flow has terminated, it uses these tables to proactively add a rule that allows the backup flow to not be redirected. The rules installed are such that the flow identifier is similar to the original flow except that the client port is ignored and the server port has changed. For example, if a flow with the flow identifier $(srcIP : 10.0.0.1, dstIP : 10.0.0.2, sport : 43823, dport : 502, prot : 6)$ crashes and the backup server is at address 10.0.0.3, the IDS will install the rules $(srcIP : 10.0.0.1, dstIP : 10.0.0.3, dport : 502, prot : 6)$ in *backupTable* and $(src : 10.0.0.3, dstIP : 10.0.0.1, sport : 502, prot : 6)$ in *backupTableRev*. These table are placed before the flow table in the P4 pipeline and if a packet matches, a cloned version is sent to the second level and the original packet is sent to its destination. At the arrival of the packet on the second level, the full usual 5-tuple rules are installed on the flow table of the switches and the rule in the two backup tables are removed so that the strict verification of the communication occurring between the MTU and the PLC becomes active.

4.3.3.3 Experiment with Backup Flow Initialization

In order to test the management of backup flows, we set up a small experiment with an MTU, a PLC, its backup, and the IDS. The switches implement the two-whitelists P4 tables as well as the table and the metadata registers to detect retransmissions. Remember that the goal of our approach is to allow the system to quickly move to a backup PLC in case of a crash and that the new flow is treated as a legitimate flow. We run the topology shown in Figure 4.14 where PLC1 (resp. PLC2) is the main PLC (resp. backup) and each link has a 10ms latency.

The MTU and the PLC1 exchange messages for 20s with a one-second period. Then the PLC stops responding, thus simulating a crash, which forces the MTU to switch to the backup PLC. The MTU will create two new flows

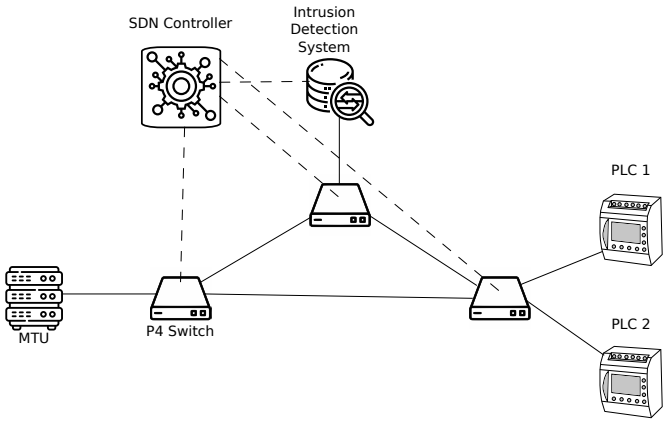


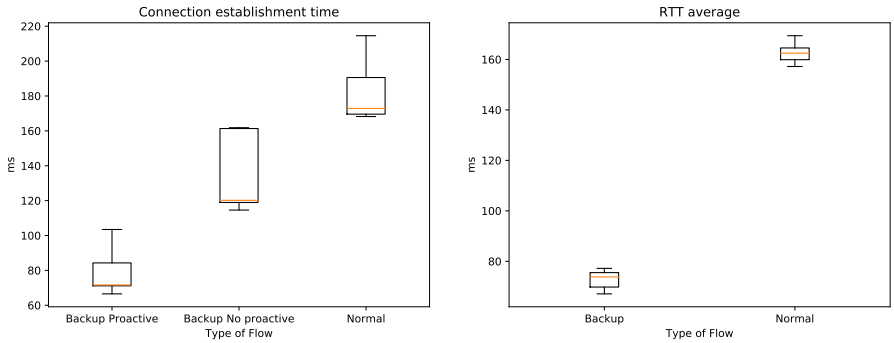
Figure 4.14: Backup flow initialization

when connecting to PLC2, the backup flow, and, to demonstrate the advantage of our approach, a normal flow not part of the whitelist. Figure 4.15a and 4.15b show the average time needed by the MTU to receive the SYN-ACK after sending the SYN packets for the two new flows.

There are two cases for the backup flows as far as the TCP establishment is concerned: The one where the second level proactively anticipates the occurrence of the backup flow and install rules accordingly and the other case where the second level waits for the first packet of the backup flow to install any rules. As it can be seen in Figure 4.15a, the proactive behavior of the controller allows for a faster establishment than the reactive behavior. With the proactive approach, The first packet (SYN packet) is directly sent to its destination and only a clone is sent to the controller thanks to the *backupTable* installed on the switches. Since there is no redirection, the packet does not experience delay. With the non-proactive approach, the first packet still has to be seen by the controller to install the correct rules for the new flow. Although slower than with the proactive approach, it is still faster than a normal flow since upon seeing the first packet of the backup flow, the controller directly installs the rules for the two unidirectional flows composing the backup flow.

However note that, while the proactive approach is more efficient, it comes with a security risk. During the crash of the main PLC, rules are installed such that any connection can be established between the MTU and the backup PLC. If an attacker is able to send packets from the MTU, it could trick the second level to think that a crash has occurred by retransmitting packets of the main flow, which would then allow the attacker to create a flow not monitored by the IDS afterward. Such capabilities are beyond our threat model since the attacker is able to control the MTU which is the worst

possible scenario.



(a) Comparison of TCP establishment **(b) Comparison of average RTT**

Figure 4.15: Backup Flow behavior

4.3.4 Discussion

The SDN-based ICS we have presented has encouraging results when it comes to the drawbacks of using a whitelist. It has some advantages but also some requirements that we discuss in the following.

IDS Load: A traditional network-based IDS monitors all the traffic in a network. Since the traffic in an ICS network is periodical and highly repetitive, most of the time the IDS monitors the same traffic again and again. This puts an unnecessary load on the IDS. With our design, the monitoring task is shared across the network and the IDS only monitors suspicious packets or critical events.

Configurability and Extensibility: Using P4 makes our design flexible and extensible. It can be extended, with minimal effort, to support other ICS protocols, such as DNP3 or any vendor-specific proprietary protocols. It is an important advantage given that it is common for some ICS to use proprietary protocol. Plus, unlike other distributed IDS , the programmability of the switches enables us to analyze protocol messages directly on the switches. Moreover, the centralized design of the IDS makes it easier to configure and update the whitelists and IDS policies for the entire network. The main limitation of our approach is that it requires that the application-layer messages analyzed by the IDS are unencrypted. This is not uncommon in ICS networks as ICS operators are still hesitating to move to protocols with encryption because they shy away from the burden of key management and the higher resource consumption on the PLCs. However, given the increasing number of attacks against ICS, it is very likely that at some point in the future ICS will

have no choice to use some level of encryption. In that case, the IDS only acts as a flow-whitelist.

Attacks Against the IDS: An attacker can attempt to perform a DoS attack against the security engine by generating a large number of random packets redirected by the switches to the engine. If the attack is successful, the IDS loses its ability to analyze and forward non-whitelisted packets. To protect the IDS, one could implement a caching mechanism between the IDS and the switches similar to the one proposed by [WXG15], although this would increase the delay for each packet. However, it should be noted that even a complete crash of the security engine does not impede the normal functioning of the network; all flows and function codes that were already whitelisted on the switches before the attack will continue working. Furthermore, since the security engine is not co-located with the controller, the network can still be manually managed.
If the capability of the IDS to add new table entries is enabled by the operator (see Section 4.3.1.2), an attacker can also attempt to fill up the flow tables [6] on the switches with new entries. To perform this attack, the attacker has to generate a large number of legitimate-looking flows with new flow signatures. We do not consider this attack as very critical since the IDS would still be able to process and forward non-whitelisted packets.
Another issue is the security of the control plane. The communication between the controller and the switch must be secured in any case. Since this is not specific to our IDS it is not discussed here.

Availability of P4-programmable Switches and Migration Costs: As described in Section 4.3.1, our design requires that P4-programmable switches are deployed in the network. P4 is still a relatively young technology and we are currently only aware of one manufacturer of such switches. Another issue is the migration cost in existing ICS infrastructures. Adding one or more traditional firewalls to an existing network is certainly cheaper than our solution which requires replacing existing switches with programmable ones. However, our solution is much more convenient to manage and also more flexible: an SDN approach allows to centrally manage the entire IDS and the P4-programs can easily be adapted to new protocols or protocol changes.

4.4 Other Benefits of SDN for ICS: An Eavesdropping Countermeasure

With the two-level IDS, we leverage SDN for intrusion detection, but the ability to program the network allows to do more. Even without middleboxes in-

[6] Filling up the Modbus tables is not possible because there are only 128 non-error function codes and the number of legitimate function-code/message-length combinations is limited in Modbus.

specting packet payload, just the knowledge about the routing can be already valuable for security. For example, in an IP spoofing attack, an attacker sends packets to impersonate a host by using its IP address. Depending on the position of the attacker with regards to the impersonated host, the packet's path through the network can be a good indication of whether or not a packet has been spoofed. Since the controller has a complete view of the network topology and the forwarding policy detecting such an attack can become possible.

In 2015, da Silva *et al.* [Sil+15] proposed to employ SDN techniques to improve the security of ICS networks. To avoid that all packets are forwarded along the same path, their multipath routing strategy periodically alternates between several paths from a source host to the destination host, such that an eavesdropper cannot capture the entire communication. This alternation is implemented by reprogramming OpenFlow-enabled switches based on rule timeouts. We observe that the proposed approach has an important drawback: It can result in large delay peaks when a network packet arrives at a switch at the moment the network is moving from one path to another. Obviously, for a time-sensitive environment such as ICS, such delay fluctuations should be avoided.

In the following, we extend the idea of da Silva and propose an approach that makes use of rule priorities in OpenFlow. We propose the priority multipath routing strategy to avoid delay peaks. We also explain the importance of selecting appropriate alternative paths and we propose to reduce the path selection process to solving a convex flow problem [AMO93].

We validate our approach by simulation experiments. Our results show that our approach significantly reduces the number of table misses and effectively eliminates delay peaks and that the selected paths are a good compromise between disjointness and cost.

4.4.1 Mitigating Eavesdropping

Network eavesdropping is a network attack where a malicious attacker sniffs a communication channel to capture network traffic. The main goal of eavesdropping is to steal confidential information. If encryption is not used or not well implemented, which is the case in many ICS protocols [SFS11][EL16], the attacker can get access to passwords or sensitive information. In ICS, the latter can include information about the type of attached devices and details on the controlled physical plant processes, their parameters, and their current state.

Da Silva *et al.* [Sil+15] propose the use of SDN to improve resilience against eavesdropping attacks in SCADA networks. Their solution enforces a multipath routing strategy: Instead of using the shortest path during the entire communication between two hosts (e.g. an MTU and a PLC) several paths

are computed and the network is instructed to periodically switch between them. In this way, an eavesdropping attacker that got access to a particular network link cannot follow the entire communication. This approach does not require changes on deployed hosts. In this section, we provide a short description of the approach proposed by da Silva *et al.* and discuss its drawback.

4.4.1.1 Multipath Routing Strategy

The multipath routing strategy relies on the `hard_timeout` timer specified in OpenFlow [Fou09]. When installing a rule in an OpenFlow switch, the controller can specify how long it wants the rule to be present in the switch. When the timer expires, the rule is removed from the switch. Optionally, the switch sends a `FlowRemoved` message to the controller to notify it about the expiration.

Figure 4.16 shows a small network with four OpenFlow-enabled switches $s1$–$s4$ and two hosts A and B. The multipath routing strategy works as follows. Initially, the flow tables of all switches are empty. The first time host A sends a packet to host B, the switch $s1$ receives the packet. Because there is no matching rule, the switch sends the packet to the controller. The controller computes k paths from A to B, i.e. the two paths $P_1 = [s1, s2, s4]$ and $P_2 = [s1, s3, s4]$ in our example, and installs two types of rules on the switches:

- *Dynamic* rules are installed on the switches that split the paths, instructing the switches to forward the packets sent by host A toward host B on path P_1. Dynamic rules have a finite hard timeout of T seconds. In our example, one dynamic rule is installed on $s1$.

- *Static* rules have an infinite timeout, i.e. they never expire. They are installed on all other switches used by P_1 and P_2 ($s2$, $s3$ and $s4$ in our example) and instruct them to forward packets toward B following the paths.

After having installed the rules, the controller tells $s1$ to forward the packet according to the rules.

The result of this strategy is that packets from A to B follow path P_1 during the first T seconds. After T seconds, the rule in $s1$ expires and is removed. The switch notifies the controller and the latter installs a new rule which instructs the switch to forward packets on path P_2. After T seconds, the rule expires again and a new rule is installed and so on. Note that the controller computes the paths only once when A sends the first packet to B. Also, note that only two paths P_1 and P_2 exist in our small example, but the approach is capable to switch between $k > 2$ paths in more complex networks.

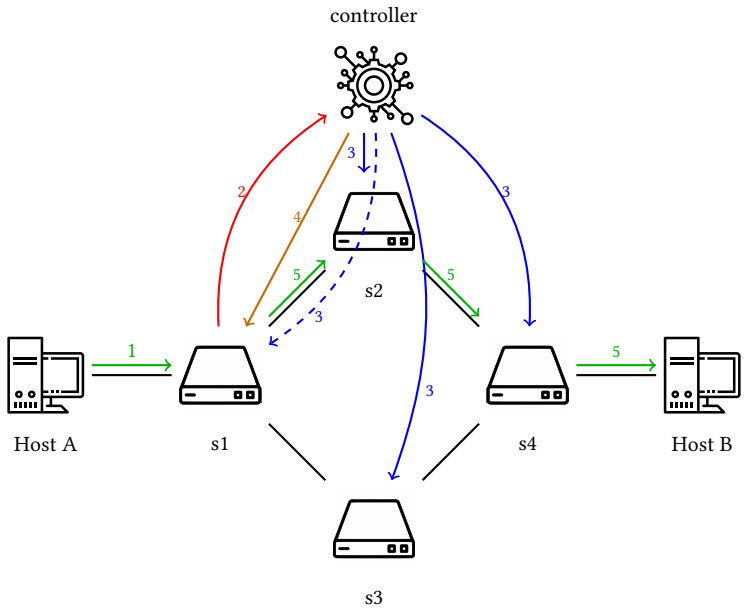


Figure 4.16: Flow installation process: (1) Host A sends a packet to B; (2) S1 has no rules, so it sends the packet to the controller; (3) Controller computes paths and installs a *dynamic rule* on S1 and *static rules* on the other switches; (4) Controller commands S1 to resend packet; (5) packet is sent to B.

4.4.1.2 Drawback

When a rule has expired for a flow from *A* to *B*, there is a period of time during which there is no matching rule in the flow table. At least a full round-trip from the switch to the controller is needed before any packet can be forwarded for that flow. This is generally not a problem since traffic in an ICS network is usually slow and regular. However, variations can occur, for example when a SCADA operator changes the rate at which a field device is queried [R R12] or simply issues a command directly to a field device. The latter scenario is particularly worrisome since the command could have been issued due to an emergency, i.e., exactly when a low-latency communication is needed most.

4.4.2 Priority Multipath Routing Strategy

This section provides a description of our approach and how it avoids the drawback identified in the previous section.

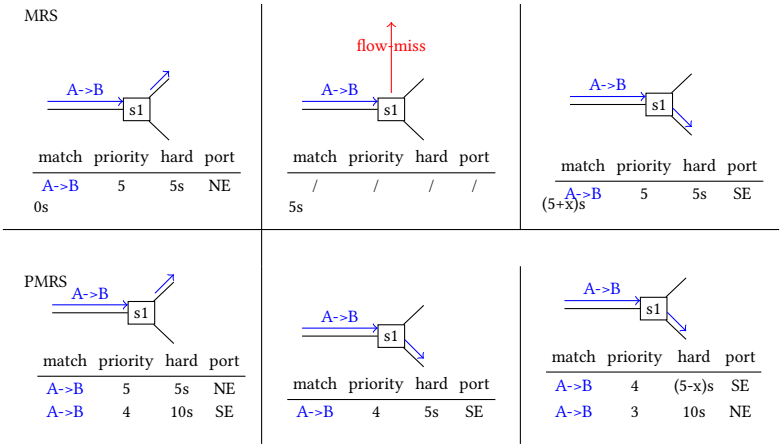


Figure 4.17: Rule expiration process: Above is the multipath routing strategy and below the priority multipath routing strategy. The state of a flow table of the switch *s1* is represented at 0s,5s and (5+x)s where x is the time for the controller to install a rule. Arrows specify the path used to forward the packet from host A to host B

4.4.2.1 Description

As previously mentioned, the issue with the multipath routing strategy exists because there is a period of time in which there is no rule for a flow. Our approach uses *rule priorities* to avoid this. Rule priorities are a feature of OpenFlow since version 0.8: Each rule of a flow table is assigned a 16-bit priority number. If several rules match the same packet, the switch will apply those actions described by the rule with the highest priority [Fou09]. Our approach differs from the basic multipath routing strategy described in the previous section in the following way: When the controller prepares the path between two hosts A and B it will install *two* dynamic rules r_1 and r_2 on the path-splitting switches such that

$$\begin{aligned} ht(r_1) &= T \\ ht(r_2) &= 2T \\ priority(r_1) &= \text{maximum priority} \\ priority(r_2) &= priority(r_1) - 1 \end{aligned}$$

where $priority(r)$ is the priority of rule r and $ht(r)$ is the value of the hard timeout for rule r.

Rule r_1 forwards packets along path P_1, while rule r_2 uses path P_2. When a packet of the flow from A to B arrives at the switch, it will match rule r_1 because of its higher priority. When r_1 expires, r_2 will immediately become the

matching rule for the flow and the path changes from P_1 to P_2. In the meanwhile, the switch informs the controller that rule r_1 has expired by sending a `FlowRemoved` message, causing the controller to install a new rule r_3 with $priority(r_3) = priority(r_2) - 1$ and $ht(r_3) = 2T$ for path P_1. Later on, when r_2 expires, r_3 will become the matching rule and the controller will install a new rule r_4 etc. Every time a rule expires, the path used for forwarding is changed.

Figure 4.17 shows the changes happening on an OpenFlow switch flow table with both the multipath routing strategy and our priority multipath routing strategy.

As can be seen, rules use decreasing priorities until the lowest possible priority value, which is 0, is reached. Our controller tackles this problem in the following way: As soon as a rule r_n with priority 1 expires and rule r_{n+1} with priority 0 becomes the active rule, the controller installs a rule identical to rule r_{n+1} with maximum priority and removes r_{n+1}.

It is worth mentioning that the number of rules for a flow present on a splitting node does not depend on the number k of paths computed; it is always two.

It should be also noted that out-of-order delivery by this approach is unlikely to happen in ICS networks. Protocols used in ICS networks such as Modbus/TCP have a rather small protocol data unit (maximum 256 bytes in Modbus/TCP [Org06]).

4.4.2.2 Periods of Inactivity

To avoid that switches and the controller exchange messages every T time unit even if there is no traffic, our approach uses the `idle_timeout` [Fou09]. In contrast to the hard timeout, the idle timeout specifies how long a rule can stay *inactive*, i.e. without any matched packets, before it is automatically removed by the switch. Our controller sets the value of the idle timeout lower than the hard timeout, so that if there is no traffic, the rule will expire because of the idle timeout. When a rule expires, the controller can determine the cause of the expiration by inspecting the `reason` field in the `FlowRemoved` message. If the rule has expired because of the idle timeout, the controller will not install a new rule on the switch, in this way avoiding unnecessary manipulations in inactive networks. However, there is a price to pay: the next arriving packet will trigger a message exchange between the switch and the controller. Note that this does not affect non-splitting switches since their rules are static.

4.4.3 Paths Selection

How alternatives paths are chosen has an impact on the mitigation of eavesdropping. The more links are shared by the paths, the larger the exposure

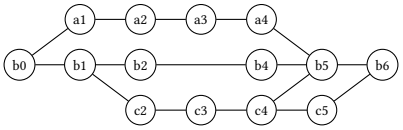


Figure 4.18: Paths selection - b0 is the source and b6 is the destination

to potential eavesdroppers. Ideally, the k paths for a pair of hosts should be edge-disjoint. At the same time, since ICS networks have real-time constraints, choosing a too long path to ensure disjointness is undesirable. The path selection must therefore find a compromise between eavesdropping risk and path length.

Consider the Figure 4.18. Suppose we want to find maximally disjoint k paths between b0 and b6. One approach to achieve this (and used in [Sil+15]) is to run Dijkstra's shortest-path algorithm k times. After each iteration, the cost of edges included in the computed shortest path is increased tenfold. In our example, this approach selects the paths [b0,b1,b2,b4,b5,b6] and [b0,a1,a2,a3,a4,b5,c4,c5,b6]. While the two paths are completely disjoint, the second path is much longer than the first. We will refer to this approach as the *Iterative Shortest Path* algorithm. Another possibility is to use the algorithm proposed in [Bha99], which ensures disjointness and that the sum of the costs of all paths is minimum. In our example, the algorithm selects [b0,a1,a2,a3,a4,b5,b6] and [b0,b1,c2,c3,c4,c5,b6], which is a good trade-off between eavesdropping risk and path lengths. The problem is that the algorithm does not work if there are no completely disjoint paths.

In this section, we describe a path selection algorithm that takes into account path lengths and their disjointness.

4.4.3.1 Minimum Cost Flow Problem

The problem of finding maximally edge-disjoint paths can be seen as an application of the minimum cost flow problem (MCP) [AMO93]. Let $G = (N, E)$ be a directed graph [7] where N is the set of nodes and E is the set of edges. In a minimum cost flow problem, each edge $(i, j) \in E$ is assigned a cost c_{ij} representing the cost by unit of flow on that edge and a capacity u_{ij} specifying the maximum flow value that can be sent on that edge. The remaining capacity of an edge when the flow has been assigned is called the residual capacity. If the residual capacity is 0, then the edge is saturated. The residual graph is the subgraph of the initial graph where all edges have a residual capacity strictly greater than 0 and saturated edges are reversed as well as their cost. An edge (i, j) with cost 1 becomes (j, i) with cost -1.

[7] This can be adapted to the undirected case by replacing an undirected edge by two edges (i, j) and (j, i).

Each node $i \in N$ is assigned an integer demand value $b(i)$. If $b(i) > 0$ then the node is a supply node and if $b(i) < 0$ then it is a demand node. If $b(i) = 0$ then it is a transient node. The goal is to assign a flow value x_{ij} to each edge (i, j) to satisfy the demand of each node (mass balance constraint):

$$\sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(i,j)\in E\}} x_{ji} = b(i) \quad \forall i \in N$$

without over-saturating edge (i,j):

$$0 \leq x_{ij} \leq u_{ij} \quad \forall (i, j) \in E$$

at a minimum cost:

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \quad (objective\ function)$$

To find k maximally edge-disjoint paths, we set the demand of the source node to k and the demand of the destination node to $-k$. All other nodes are transient. We limit the capacity of each edge to k. The amount of flow x_{ij} on the edge (i, j) represents the number of paths that use this edge. If the capacity u_{ij} is set to 1, it means that the edge can be used only once which would give the same result as the algorithm in [Bha99]. By doing this, we address a common assumption of MCP stating that all $u_{i,j}, b(i), c_{i,j}$ are integral. We also assume that edge costs are non-negative.

4.4.3.2 Reduced Cost

Before describing the algorithm to compute the paths, we have to explain the reduced cost. Each node i is assigned a number $\pi(i)$ denoting its *potential*. It can be interpreted as the cost of obtaining one unit of flow at node i. The reduced cost of an edge (i,j) is $c^{\pi}_{ij} = c_{ij} - \pi(i) + \pi(j)$.

An optimality condition of MCP is that the residual graph does not contain negative cost cycles. The intuition behind is that if the residual graph contains a negative cost cycle then the objective function can be improved by pushing flows along the cycle. The condition is equivalent to the *reduced-cost optimality condition* [AMO93] which states that a solution for a minimum cost flow problem is optimal if and only if some set of node potentials π satisfy the reduced cost optimality conditions:

$$c^{\pi}_{ij} \geq 0 \quad \forall (i, j) \in E \quad \text{in the residual graph.}$$

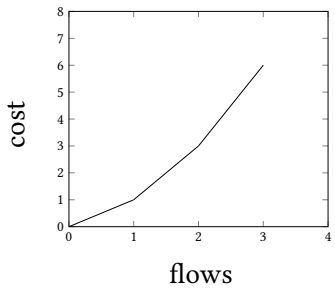


Figure 4.19: Piecewise linear cost function with $u_{ij} = 3, (i, j) \in E$

4.4.3.3 Capacity Scaling Algorithm

To achieve disjointness, we use an adapted version of MCP called *convex flow problem* [AMO93]. In MCP, the cost flow of an edge varies linearly with the amount of flow on that edge. Each unit of flow added to an edge (i, j) increments the objective function by c_{ij} independently of the current amount of flow on (i, j). With the convex flow problem, the cost of an edge is a convex function $C_{ij}(x_{ij})$, so that the cost per unit of flow on (i, j) will increase as x_{ij} increases. A convex function makes the algorithm avoiding using the same edge several times if it is not necessary. If an edge has been used once, using it a second time will have a bigger impact on the objective function than using it the first time, however without completely prohibiting it.

To solve the convex flow problem, we first transform it into a minimum cost flow problem using the method proposed by [AMO93]. We assume that cost functions are piecewise linear convex. A piecewise linear function is a function whose graph is segmented into straight lines of varying slopes. The slopes of the segments represent the different costs that an edge can have. For example, with the cost function on Figure 4.19, an edge can have a cost of 1,2 or, 3. The transformation consists of replacing the edge on the initial graph with s edges where s is the number of segments of the cost function. The capacities of the new edges are equal to the absolute differences of two consecutive breakpoints. Considering the example in Figure 4.19, we would get three edges of capacity 1 and with respectively cost 1, 2, 3.

The algorithm we use to solve the resulting minimum cost flow problem is the *capacity scaling algorithm* [AMO93]. The complexity of the algorithm is $O(m \log US(n, m, nC))$ where m is the number of edges, n the number of nodes,U is the supplier/demand upper bound (in our case k) and $S(n, m, nC)$ is the time required to solve the shortest path problem with nonnegative cost edge whose cost is greater than C. The capacity scaling algorithm uses a pseudo-flow x_{ij} on each edge (i, j). The difference between a flow and a pseudo-flow is that the pseudoflow does not have to satisfy the mass balance

constraint. It uses rather an imbalance value $e(i)$ defined as:

$$e(i) = b(i) + \sum_{\{j:(i,j)\in E\}} x_{ij} - \sum_{\{j:(i,j)\in E\}} x_{ji} \quad \forall i \in N$$

If $e(i) > 0$, $e(i)$ is the excess of node i, if $e(i) < 0$ then $e(i)$ is the deficit of node i, otherwise node i is balanced. The algorithm shifts from the pseudoflow to a flow by augmenting iteratively flows on shortest paths between nodes with excess and nodes with deficit while satisfying the reduced cost optimality condition.

To reduce the number of steps, the algorithm tries to pushes as much as possible amounts of flow on the shortest path with the highest bottleneck. It does so by maintaining a scaling parameter Δ. The algorithm ensures that each push contains exactly Δ flow. If it is not possible to push Δ flow because there is no excess node of a least Δ nor a deficit node of at least Δ, the algorithm divides Δ by 2 and repeats the process. The algorithm works on the Δ-residual graph $G(x, \Delta)$ which is a subgraph of the initial graph but where all edges have a residual capacity of a least Δ. At some point, Δ is equal to 1 and if there is no more push possible, it means there is no more excess or deficit so the mass balance constraint is satisfied and the solution is a flow and it is optimal because it satisfies the reduced optimality condition.

4.4.4 Experimental Results

This section first describes the experimental setup. Then, it presents and discusses the results.

4.4.4.1 Methodology

We use the POX controller [20b] to implement our approach since it provides useful components to perform certain tasks such as topology discovery using the Link Layer Discovery Protocol [APF15]. We use Mininet as a network simulator [LHM10b]. We simulate the network shown in Figure 4.20, which is similar to the one studied in [Sil+15]. The controller calculates $k = 3$ paths and all links have a convex cost function C with

$$C(x) = \begin{cases} x, & 0 \leq x \leq 1 \\ 2x, & 1 < x \leq 2 \\ 3x, & 2 < x \leq 3 \\ 4x, & 3 < x \leq 4 \end{cases}$$

A path will have a lifetime of $T = 5$ seconds before it is changed. We simulate a 30ms delay between the switches and the controller using the `netem` tool [21a]. This represents a (geographically) large SCADA network. Simulations run for ten minutes.

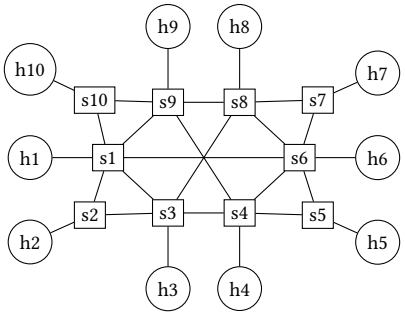


Figure 4.20: Simulated network. Rectangles are switches and circles are host.

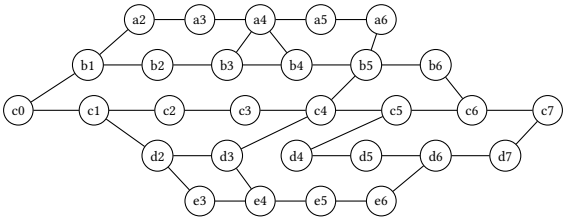


Figure 4.21: Paths selection - c0 is the source and c7 is the destination

We compare the basic multipath routing strategy (MRS) from [Sil+15] with our priority-based approach (PMRS). We use two simulation setups:

Setup 1: To measure table misses, which occurs when no matching rule is found by a switch for an incoming packet, the host h4 acts as a UDP server. h10 sends UDP packets to h4 for ten minutes with 25ms between each packet.

Setup 2: To measure delays, the host h10 acts as an MTU polling the host h4 which acts as a PLC. The MTU sends a Modbus TCP request every 2 seconds and the PLC responds. To simulate an operator manually sending commands to the PLC, the MTU will send every second with a probability of 10% a burst of five Modbus TCP requests.

We also compare the *Capacity Scaling* (CP) algorithm and the *Iterative Shortest Path* approach (ISP) described in Section 4.4.3 on the graph presented on Figure 4.21. We compare the packet exposure average for each edge, and the total sum cost for all paths. The packet exposure is the percentage of traffic that an attacker could obtain by eavesdropping on a link. Each value is compared from 1 to 5 paths. We do not include the algorithm from [Bha99] as it requires that there exist at least k completely disjoint paths.

Finally, we apply both algorithms on random geometric graphs (RGGs) to find 7 paths. A random geometric graph is an undirected graph created by placing nodes in some space such that two nodes are connected if their distance is below a given value. We generate 100 RGGs of 120 nodes. Then we add a source node and we attach it to 3 nodes on the generated graphs.

Table 4.2: Packet exposure average for Figure 4.21

Algorithm	Packet exposure average				
Number of path k	1	2	3	4	5
Capacity Scaling	100%	50%	39.13%	33.09%	32.15%
Iterative Shortest Path	100%	50%	37.99%	33.97%	33.14%

Table 4.3: Sum cost for all paths in Figure 4.21

Algorithm	Sum cost for all paths				
Number of path k	1	2	3	4	5
Capacity Scaling	7	17	24	38	44
Iterative Shortest Path	7	20	30	40	49

We do the same for the destination node. Each edge has a cost of 1, and the piecewise linear function is the same as the one presented earlier.

4.4.4.2 Results

Table 4.2 shows the average packet exposure on the links when the controller computes 1 to 5 paths. It is a measure of the amount of traffic an attacker is able to get from eavesdropping on communications between $c0$ and $c7$. There is not a big difference in terms of packets exposed between the two approaches meaning that both algorithms are able to compute disjoint paths. However, the capacity scaling algorithm uses a more global view of the paths so that they do not differ from each other and minimize the overall cost.

Table 4.3 shows the sum of the cost for all paths and Table 4.4 shows the average cost and variance of all the paths. As can be seen, CP not only selects shorter paths but also paths that are less diverse length-wise which is preferable as it impacts the delay seen by the sender. This results is also

Table 4.4: Average and Variance of cost for all paths in Figure 4.21

Algorithm	Average, Variance of cost for all paths				
Number of path k	1	2	3	4	5
Capacity Scaling	7, 0	8.5, 0	8, 0	9.5, 3	8.8, 2
Iterative Shortest Path	7, 0	10, 9	10, 6	10, 4	9.8, 4

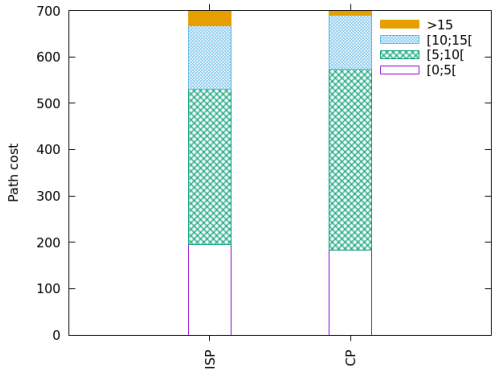


Figure 4.22: Path cost repartition

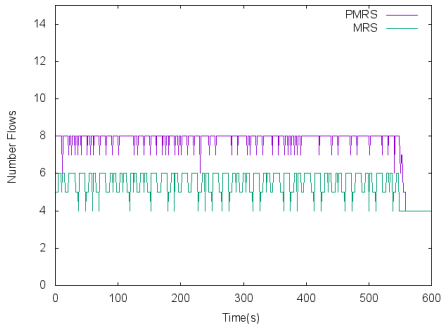


Figure 4.23: Number of rules in the network (setup 1)

shown in Figure 4.22 which shows the repartition of all the paths selected according to their cost on the RGGs. It shows the number of paths whose cost is strictly less than 5, between 5 and 10, etc. The plot shows that CP has more short paths. ISP has *33* paths that are more expensive than 15 while the CP algorithm only has *8*. Also, by observing selected paths, we noticed that the CP algorithm selects paths such that the cost of the paths does not vary too much from each other while paths selected by the ISP approach have more disparity in terms of cost. If costs are assigned according to delay, the former property means a lower jitter, which is desirable in a real-time environment.

The number of table misses for setup 1 is *166* for MRS and *11* for PMRS. In MRS, this number is higher because of the issue described in Section 4.4.1.2.

Figure 4.23 shows the evolution of rules installation during the simulation. There are more rules in PMRS as there are always two rules on splitting nodes instead of one. The maximum number of rules for a flow in MRS is equal to n where n is the number of switches in all paths. In PMRS, it is equal to $2 \times m + (n - m)$ where m is the number of splitting switches. We consider

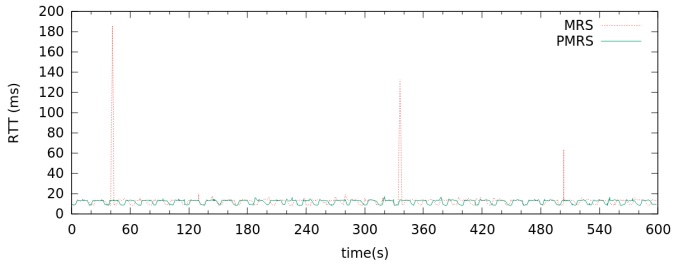


Figure 4.24: Delay over time (setup 2)

this overhead to be acceptable since it avoids high delays in a time-sensitive environment.

Figure 4.24 shows how the round trip time (RTT) between h4 and h10 varies over time in setup 2. We can see that the RTT can have large peaks when MRS is used. Those peaks occur when packets belonging to a burst arrive at a switch but there is no matching rule installed. The delay the packets suffer will depend on the time of their arrival. If they arrive at the switch immediately after the dynamic rule has expired, the delay will be larger than when they arrive shortly before the next dynamic rule is installed. Table misses can also occur on s1 as it is a splitting switch for paths from h4 to h10, so a Modbus TCP request can encounter a table miss several times. Naturally, there are no such delay variations with PMRS. Note that the ripply shape of the baseline is caused by the different lengths of the three paths.

4.5 Conclusion

In this chapter, we expanded our discussion on Flow-based IDS to whitelisting techniques. The idea behind whitelisting is simple but its application can be become quite complex in real networks. Creating a perfect whitelist requires having perfect knowledge of the system, which is difficult to achieve. In an ICS systems, like many other systems, some of the hardware and the software is not necessarily managed by the owner of the system and involves third parties (e.g., manufacturers or vendors) that provide enough information on how to use a specific product but not necessarily on how it works. This is problematic for the whitelist as the equipment can perform actions that are legitimate but unknown to the users, like a PLC pulling updates from the Internet.

To facilitate the use of Flow-Whitelisting in ICS systems, we proposed to use SDN. It eases the management and control of the network by transferring the handling of the control plane from network devices to a controller. This centralized approach has some advantages that ICS could benefit from,

notably the management of the whitelist. In SDN, the network devices are only responsible for forwarding packets, whereas the routing decisions are taken by the controller. We leverage this property by installing the whitelist on the networking devices and use a second level to allow a real-time modification of the whitelist. Since the whitelist might not be complete, being able to add missing flows that are legitimate to the whitelist is a necessity. To decide whether a non-whitelisted flow should be added to the whitelist, we presented a series of checks and validations that the flow must pass.

We also presented a way on how ICS can benefit from SDN other than using an IDS. We improved the eavesdropping mitigation technique found in the litterature by showing its drawbacks and fixed it.

Like for any IDS, our approach is not effective if an attacker knows how to create flows such that they pass those checks. One could decide to use an anomaly-based detection algorithm which can detect new attacks but also results in more false positives. In the worst case, attackers could completely bypass the IDS if they performed attacks during the capture of the traffic used to create the original whitelist. Having a PLC at the mercy of an attacker is highly problematic given the nature of ICS systems. Since there is usually no authentication, the attacker can influence the behavior of the PLC by sending valid commands. If the attacker has some knowledge of the physical process being monitored and controlled by the ICS system, they can cause serious damages that can go beyond the cyber-part of the system. In the next chapter, we will discuss what can be done to handle such a situation and propose a solution to detect when an attacker is tampering with a PLC in such a way that it damages the physical process.

Detection of Process-Oriented Attacks

5

In the previous chapter of this thesis, we focused on detecting attacks targeting the networking infrastructure of an ICS system. With that work, we were in good company in the scientific community: indeed, a large part of the published work on intrusion detection for ICS attempts to detect attacks by looking at the communication between the various ICS components. They monitor general characteristics of the ICS network and its hosts, for example in order to detect flooding attacks [ZJS11], or they monitor details of the messages exchanged between the control servers and field devices in order to detect abnormal messages or message sequences [GW13]. This approach has the advantage that it is relatively general and does not require knowledge of the physical process.

However, as we have repeated several times, ICS systems include both a cyber part and a physical part that more or less directly interacts with the world. Therefore, one of the threats that ICS systems face are attacks that target the physical process controlled and managed by the system. Such attacks are called *process-oriented attacks* and they usually try to disrupt or completely stop the physical process. Process-oriented attacks can be perpetuated through cyber means, for example by infecting PLCs and issuing well-formed but undesired command sequences to actuators. In a perfect world, security measures would completely prevent such a scenario to occur but there have been many cases where a simple phishing attack could help an attacker to achieve their goal [Ant+17]. For that reason, researchers have also proposed *process-aware* detection approaches where the detection system looks for any behavior violating the semantics of the physical process controlled by the ICS.

Process-aware IDS monitor *process variables*, i.e. the sensor values and the actuator states in the physical process. Among the popular approaches for process-aware IDS, two broad classes can be distinguished. Specification-based IDS compare the behavior or state of the monitored physical process to a specification provided by the process designer. Such IDS work especially well for processes where a formal specification is available or can be easily obtained. A typical example are power grids, for which the physical laws governing the power flows are well understood [FCR19; Lin+18; CMW19]. However, in this chapter, we are interested in ICS where a formal specification

or digital twin is not available for various reasons, for example because the physical process grew "organically" over its life time without its documentation being updated. For such processes, IDS designs have been proposed where the specification is *learned* from historical data. Again, two major categories can be identified among those IDS found in the literature: The first one leverages learned predictive models to predict the values of the process variables and the IDS raises an alert if the difference between the predicted value and the measured value is greater than a threshold [Had+14; Urb+16]. The drawback of this approach is that it does not fit well to the noisy nature of real sensors which makes comparing values challenging. The second category uses invariant rules that must be always satisfied [Kou+16; AM16; Fen+19]. The drawback of Invariant-based IDS is that they treat the states of the physical process without a notion of evolution in time. An attacker can evade detection by exploiting this limitation, for example by slowing down or permanently locking the physical process in a valid state, as demonstrated in this chapter.

To overcome the above problems, in this chapter we present a process-aware IDS that leverages the temporal characteristics of the system states. The IDS compares the current behavior of the ICS against a learned model that describes allowed transitions between critical values of the process variables and their time properties.

The structure of the chapter is as follows. Section 5.1 presents the concept of process variables. Section 5.2 discusses existing work. Section 5.3 describes our detection approach. Its evaluation is presented in Section 5.4. Limitations are discussed in Section 5.5. The chapter is concluded in Section 5.6.

5.1 Physical Processes and Variables

A process variable stores a certain value of a physical process, e.g the level of water in a tank or the flow rate of a pipe. The state of a process at a given time is determined by the values of its process variables. The latter can be divided into two categories. *Discrete variables* have a small set of possible values. An example is a variable representing the state of a valve: "Open" or "Closed". *Continuous variables* have a more or less continuous value range as the range of values depends on the accuracy of the sensors. They usually represent physical properties like the water temperature measured by a sensor.

The ICS controls the process by manipulating actuators depending on sensor values. For example, once the water level in a tank has reached a certain threshold, the ICS may decide to open a valve. We refer to these values as *critical values*. Consider the physical process in Figure 5.1. The goal of this process is for *Tank 3* (T3) to release 60 units of a product C. T3 has a maximum capacity of 80 units. The tanks *Tank 1* (T1) and *Tank 2* (T2) contain

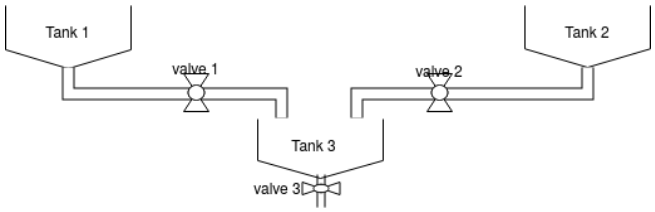


Figure 5.1: Simplified Three Tank Systems

respectively product *A* and *B*. First, T1 releases 20 units of product *A*, and then T2 releases 40 units of product *B*, so they can be mixed in T3 and create product *C*. There are six process variables in this example: one discrete variable for each valve and one continuous variable for each tank's level. The ICS has to open the different valves according to the level in T3, therefore there are three critical values for T3's level leading to a change in the valve states: 0, 20, 60. It is important to mention that actuators also have critical values. For example for a valve, there are two critical values, open and closed.

In a process-oriented attack, the attacker tries to disturb the physical process. For example, an attacker could send a command over the network to open the valve of T3 while the tank is being filled [ILW06]. An Invariant-based IDS can detect this attack because it violates the invariant of the process stating that the valve of T3 is always closed when the tank is being filled. However, the attacker could also wait until the tank is empty and then prevents the valve from closing, for example by overwriting the close command sent by the ICS. An empty tank with an open valve is a legitimate state of the system; it happens for a short time at the end of emptying the tank. The key here is the time aspect. Our IDS is based on the observation that critical values are associated with a certain duration.

5.2 Existing Process-Aware Intrusion Detection Systems

Process-oriented attacks target the physical process controlled by the ICS. This requires a certain knowledge of the process by the attacker in contrast to, for example, a DDoS attack against an ICS server. IDS that have knowledge of the physical process and use that knowledge to detect attacks are called *process aware*. Similarly to the detection of network attacks, they usually belong to the class of Anomaly-based IDS, i.e. they monitor a system and compare its activities with a model describing its expected behavior thus allowing the detection of new attacks. In this context, the model represents characteristics of the physical process being managed by the ICS. During a design or training phase the model is built and configured according to a set of characteristics of interest and then during the detection phase, a validity

test that applies the model to a specific representation of the current state of the physical process is performed to look for any abnormal behavior.

5.2.1 Invariant Models

A physical process follows a set of rules that the system maintains by manipulating the actuators. For example, when a tank is filled so that it reaches a certain limit, a valve will open. Such rules are defined by the operator to achieve some goal related to the nature of the physical process. A common approach to detect process-oriented attacks is to check whether or not the physical process is violating those rules. These rules are the invariants of the physical process, meaning that they are supposed to always be valid and when it is not the case, it is an indication that an error or attack has happened.

In [Fen+19], the detection model leverages the invariant properties of the process to detect attacks. The IDS creates invariant rules of the form $A \rightarrow B$, where A and B are predicates/conditions on the physical process. An invariant is violated when A is satisfied and B is not. There are two types of predicates: distributed-driven and event-driven. The first type considers changes in sensor readings to be determined by the current state of the physical process. The IDS assumes that each update of a sensor value follows one of K hidden Gaussian distributions. The distributions are found by fitting a Gaussian Mixture Model to the difference between each update of the values of a sensor. The predicate is satisfied for an observed sensor update if there is a high probability that the update comes from one of the distributions. The second type of predicates uses the changes of the discrete actuator states to find critical values of sensors. These changes are called events. Each time an event happens in the learning trace, the IDS attempts to predict the value of each sensor with a regression model using the other sensors as features. A correct prediction generates two predicates representing the sensor value before and after the event. After generating the predicates, the IDS uses data mining techniques to find frequent itemsets where the predicates are considered as items. This yields predicates that are often satisfied at the same time. The IDS uses them to generate invariant rules.

Other IDS, for example [AM16; AM17; Car+11], rely on invariants created from the design specification of the physical process, which means that their detection performance is limited by how accurately a human operator has described the physical process. The advantage of [Fen+19] is that the model is learned from a dataset collected on the MTU and is automatic. However, like any learning approach, it requires that the dataset is sufficiently representative of the normal behavior of the physical process and does not include any attack.

5.2.2 Predictive models

Another popular approach for detecting process-oriented attacks is the use of predictive models and time series forecasting techniques. The idea is to predict the future values of the process variables and compare the actual values with predictions. The residuals, the differences between the predictions and the observed values, are then computed and a statistical test is performed on the distribution of the residuals to check whether or not it is similar to the one observed with the validation set.

An example of such an approach is the work by Hadžiosmanović *et al.* [Had+14]. Their IDS classifies process variables in three categories according to the number of different values it observed in the learning trace. Process variables are constant if they hold only one value, discrete if they hold a few different values, and continuous otherwise. In the learning phase, the IDS stores for constant and discrete process variables all their values. For continuous variables, the IDS creates for each of them an autoregressive model

$$x_t = \phi_0 + \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_t x_{t-p} + e_t$$

of order p, where x_t is the value of the process variable at time t, $\phi_1, \ldots, \phi_p$ are determined coefficients, and e_t is a normally distributed error term with zero mean and non-zero variance σ^2. The Burg's method is used to estimate the coefficients [Hoo95] and the Akaike Information Criterion is used to estimate the order of the model [Sug78]. Once the model is computed, it is tested against a training set and the IDS stores the mean and the variance of the residuals. In addition to that, the IDS uses a complementary strategy called the Shewart Control Limits to detect deviations of the timeseries from its operational limits. The limits are calculated as values that are three standard deviations from the estimated mean.

In the detection phase, the IDS performs different kinds of checks. For constant and discrete variables, it looks if values are part of the set of values seen in the learning phase and alerts are raised when it is not the case. For continuous variables, the IDS raises alerts if (i) if the value of a variable is outside of the control limits or (ii) if the value deviates from the prediction model. For the second case, the IDS compares the residual with the prediction error during the detection phase. A significantly higher prediction error variance compared to the residual variance means that a deviation occurred from the computed model. The detection of the deviation is performed via an F-test of variance equality.

5.2.3 Other Techniques

In [CZK15], Markov chains are used to detect malicious sequences of commands sent to actuators and sensors, denoted as *sequence attacks*. States in

the chain represent events, for example, the response of a field device to a request from the server and the transitions are actions (e.g. Modbus function calls) leading from one state to another with a certain probability. The Markov chain is learned during a training phase from network communications between the control server and the field devices. At runtime, the IDS compares the current system behavior with the stochastic behavior described by the Markov chain. This approach is sensitive to the training data. If the training set is too small, it can result in over-fitted models that do not tolerate small deviations that can be often found in real systems (the problem of naturally occurring deviations has been addressed in the design of an IDS described in [BSP16]). If the training set is too large, a very general model is obtained where most transitions have a non-zero probability.

Koucham *et al.* [Kou+16] also propose an IDS to detect sequence attacks. The IDS takes specifications given by *Linear Temporal Logic* formulas that state the natural order of events in the physical process. The formulas are generated from network traces and then filtered to reduce their number. However, as the authors acknowledge, with the number of specifications they have, when an attack occurs, there are so many specifications that are violated that it is challenging to do a precise analysis. Also, the IDS has to keep a subset of the current trace to see if a specification is violated or not. A large subset has more chance to detect attacks but requires more resources, while a smaller set may miss attacks.

Sicard *et al.* [SZF18] look for system states that should never happen. They implement a filter placed between the PLC and the attached actuators and sensors. First, they identify the system states that are critical, i.e. the ones that can damage the system. Then, they create a process model using Deterministic Finite Automata, which describes all possible states and actions of the system. In addition, they create a control model, using Petri Nets, representing the scheduling of the actions taken by the system to achieve its goal. To detect attacks, they use the two models to calculate the distance between the current system state and the critical states. If the distance to the closest critical state is below a defined threshold, commands sent by the PLC to the actuators are immediately blocked. We argue that being able to trace the commands exchanged between a PLC and the actuator/sensors would require significant modifications in the field devices used in practice. Secondly, the blocking is completely hidden to the control server, meaning that it cannot differentiate between blocked commands and commands not correctly executed due to a malfunction. Similarly, Carcano *et al.* [Car+11] describe a formal language to declare critical system states and define a distance function to detect whether the current state of the system will reach a critical state in the near future. Unfortunately, they do not provide an experimental evaluation, but like similar approaches, it suffers from the fact that attacks causing

damage to the system without leading to a critical state cannot be detected.

The above approaches do not consider time, in the sense that they do not reason on how long a system should stay in a certain state or how long a transition between two states should take. There exists work that considers time in their detection process, although on a different level [LNA17; LN19; SEG18]. For example, Lin *et al.* [LNA17; LN19] leverage temporal properties of SCADA systems to detect attacks. However, their approach focuses on the temporal properties of events at the network traffic level, i.e, when network messages are sent or received. Those works are effective in detecting network attacks such as drop, replay or delay attacks however they are not looking at the behavior of the system at a process level, i.e., they are not process-aware. Yang *et al.* [Yan+20] present an IDS exploiting the correlation between the sensor readings and the commands sent by the system to identify set points. The correlation is learned from logs and represented as a control graph where the nodes can be set-points, sensor readings, or commands, and the edges are weights describing the correlation between two nodes. During the detection time, the IDS uses a sliding window to monitor the evolution of the correlation between the nodes, specifically between the sensor readings and the commands, to detect anomalies. When the computed correlation repeatedly deviates too much from the expected one according to a pre-defined threshold, an alert is raised. In that approach, time is implicitly considered because of the sliding window. This IDS is able to detect attacks rewriting the sensor reading in an obvious way, i.e., when the reading does not make sense in the state in which the process is, or in a stealthy way where the readings sent by the attacker are just the repetition of what they previously observed. The performance of the IDS highly depends on the size of the sliding windows and the value of the threshold. They influence the number of attacks detected and the number of false alerts. However, in order to choose those values, a priori knowledge of the process is required. No algorithm to select those values is provided.

5.3 A Time-based Intrusion Detection System

We present our Time-based IDS that exploits the temporal properties of process variables. The IDS learns these properties for a physical process from trace files that were recorded, for example, at the control server of the ICS (or in the network if the messages exchanged between the ICS components are not encrypted [Org06]). Once the temporal properties have been learned, the IDS can move to the detection phase where it operates on live data obtained from the same source. Our solution has been designed for ICS systems where a centralized server monitors and controls the physical process.

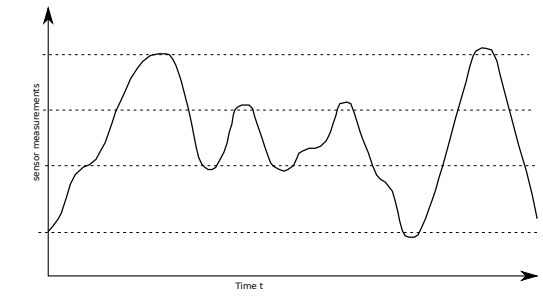
5.3.1 Scope and Assumptions

We want our IDS to be of use without too much knowledge of the physical processes. We assume that a formal specification of the physical process is not available and therefore has to be learned. The purpose of an ICS is to control and manage physical processes; it defines set points for the different process variables and ensures that the process stays within these points. We assume that process variables move between critical values. We call the event of a variable moving from one critical value to another a *transition*. For a discrete variable, like the state of a switch, a transition is quite fast, but for a continuous variable, it can take several minutes or hours depending on the underlying physical process. We assume that process variables behave consistently in their transitions throughout the lifetime of the physical process. In other words, we assume that critical values will be recurrently visited by the corresponding variable. We refer to those variables as *recurrent variables*. Such variables hold specific values when there is a change in the process state handled by the system. A recurrent variable does not necessarily need to follow a fixed order when passing through a series of critical values — only the fact that one or more critical values are recurrently visited matters. It should be noted that not all process variables behave like this. Figure 5.2 shows examples of the two types of variables. The recurrent variable in Figure 5.2a moves between specific values, represented by the dashed lines, although not necessarily with constant time between the changes. The decreasing and the increasing phases occur because the variable has reached a certain value causing a change of state. The variable shown in Figure 5.2b does not depict such a behavior. Our IDS can only monitor those variables in an ICS that behave like the one in Figure 5.2a.

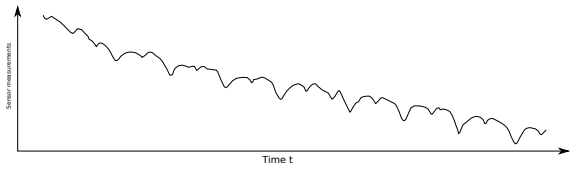
Furthermore, we assume that actuators are represented by discrete variables that hold their state as set by the ICS. That does not necessarily mean that an actuator cannot have a continuous physical state. Obviously, for example, the real speed of an engine is not a discrete quantity. Rather, it means that the ICS sets the target speed of the engine in form of discrete values.

We also assume that if an attacker is tampering with a process variable, it will have an impact on how fast it moves between its critical values or on other process variables. It may take more or less time than expected or a transition happens that was not observed during the learning phase.

Finally, we assume that the learning dataset is of sufficient length, which means all recurrent variables have visited all their critical values several times during the measurement period. Because we follow a data-driven approach, a representative trace is required, which is an obvious limitation inherent to all data-driven approaches, however getting such a trace is feasible in ICS environments as traces are consistently stored for different reasons, e.g. di-



(a) Recurrent



(b) Non-recurrent

Figure 5.2: Types of process variables

agnostics or quality assurance.

5.3.2 Attack Model

In the following, we describe the attack model considered in this chapter. An attack model defines which part of the system is targeted by an attacker, what are their goals, and what are their capabilities. We will only consider attacks targeting the physical process. The goal of the attacker is to cause damage by manipulating the process variables such that the normal operation of the process is disturbed. For example, the attacker could prevent a tank from emptying and thus cause an overflow. We assume a scenario where an attacker is inside the ICS network:

- The attacker can connect to the PLCs in the network.
- The attacker can read and write the registers of a PLC by sending requests to it.
- The attacker knows the physical process being controlled by the ICS, meaning that the attacker knows the relations and dependencies existing between the process variables and exploits that knowledge.
- The attacker knows which process variable is handled by which PLC.
- The attacker cannot forge the response of the PLC. That means that they can not control the response sent by a PLC upon receiving a request from the control server.

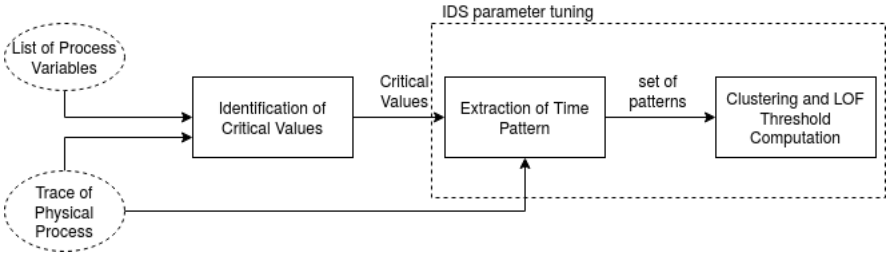


Figure 5.3: Learning phase steps

- The attacker does not communicate with the control server.

Given that model, an attacker can damage a physical process efficiently. They have enough knowledge to identify the process variables that have to be manipulated in order to achieve their goal. Since they knows the mapping between those variables and the PLCs, the attacker can send commands to them to control the process according to their goal. In this model, the control server and the system logs remain a reliable source of information and they will be used by our Time-based IDS to detect process-oriented attacks.

5.3.3 Learning Phase

The learning phase of the IDS expects an attack-free trace of the system. The trace is a series of snapshots of the process variables taken every second (this is a typical update frequency in ICS [Fen+19]).

In addition, we tell the IDS which variables are discrete and which are continuous, and which discrete variables are representing actuators.

Analyzing the relationships between potentially hundreds of process variables is a complex task. We follow an efficient approach with a minimum number of configuration parameters in the following. The IDS performs two major steps during the learning phase (Figure 5.3):

5.3.3.1 Step 1: Identifying Critical Values

For a discrete variable, the critical values are the values that it can take. In particular, an actuator corresponds to a discrete variable $a \in \mathbb{N}$. The value taken by a represent the state of the actuator. For exemple, for an actuator acting like an ON/OFF switch, $a \in \{0, 1\}$ for the ON and OFF states. In general, $C_a = \{c_1^a, \ldots, c_n^a\}$ denote the critical values of a which are the values it can take.

For a continuous variable $s \in \mathbb{R}$ holding a sensor value, its critical values are those values that cause the ICS to change the state of one or several actuators.

Let a be a discrete actuator variable and s be a continuous sensor variable. For a transition $c_i^a \rightarrow c_j^a$ from critical value c_i^a to c_j^a of a with $i \neq j$ and happening at times $t^1, \ldots, t^m$ in the learning dataset, we look at the values $v_1, \ldots, v_m$ that s takes at those times. If a certain sensor value v is linked to the transition $c_i^a \rightarrow c_j^a$, we should see $v = v_1 = \ldots = v_m$ and we conclude that v is a critical value of s (in theory, an actuator transition could be linked to two or more critical values of the same sensor s, but we have not seen that happen in practice). According to [Fen+19], sensor values can be perturbed by noise following a (doubly truncated) normal distribution with mean 0, therefore if μ is the mean of $v_1, \ldots, v_m$, we should observe that all v_i fall in a bin $[\mu - \frac{\Delta}{2}, \mu + \frac{\Delta}{2}]$ for some bin width Δ.

We repeat this for all transitions of all actuators and obtain for s a set of bins. We expect that the widths of some bins are relatively small for those actuator transitions where s has a critical value and relatively large when the value of s is not related at all to the transition. To identify to which of those two groups each bin belongs, we normalize the bin widths by the respective bin means and use *Mean Shift* [Yiz95], a non-parametric clustering algorithm for mode seeking. We then only keep the bins assigned to the cluster with the smallest mean bin width. Those selected bins $C_s = \{[l_1, r_2], \ldots, [l_{c_s}, r_{c_s}]\}$ represent the c_i^s critical values of s. We say that s has reached its ith critical value if its current value v falls in the range $[l_i, r_i]$. If multiple bins contain v, the closest is selected. Note that we can see discrete variables as variables where all the bins only contain single values, i.e. $l_i = r_i$. For continuous variables, we never let the bin width go below six standard deviations of the values observed in the learning data.

For efficiency reasons, we then discretize all bin boundaries to multiples of the smallest bin size in C_s. This introduces a small error but allows to implement the check whether a sensor has reached a critical value as a table lookup.

5.3.3.2 Step 2: Time Pattern Extraction

Once the critical values of all process variables have been identified, we extract for each continuous variable s the temporal properties of the transitions occurring in the physical process. The series of values $v_1, v_2, \ldots, v_{n-1}, v_n$ observed for a process variable s form a transition if v_1 reaches some critical value c_i^s and v_n reaches a different critical value c_j^s with $i \neq j$ and the intermediate values $v_2, \ldots, v_{n-1}$ do not reach any critical value. We call t the transition time, i.e., the number of 1-second steps the transition took, and $u = \frac{v_n - v_1}{n-1}$ the mean update step. Intuitively, an update for a sensor is the difference of two successive measurement readings by a sensor, and the mean update step is the average of all updates that were necessary to complete the

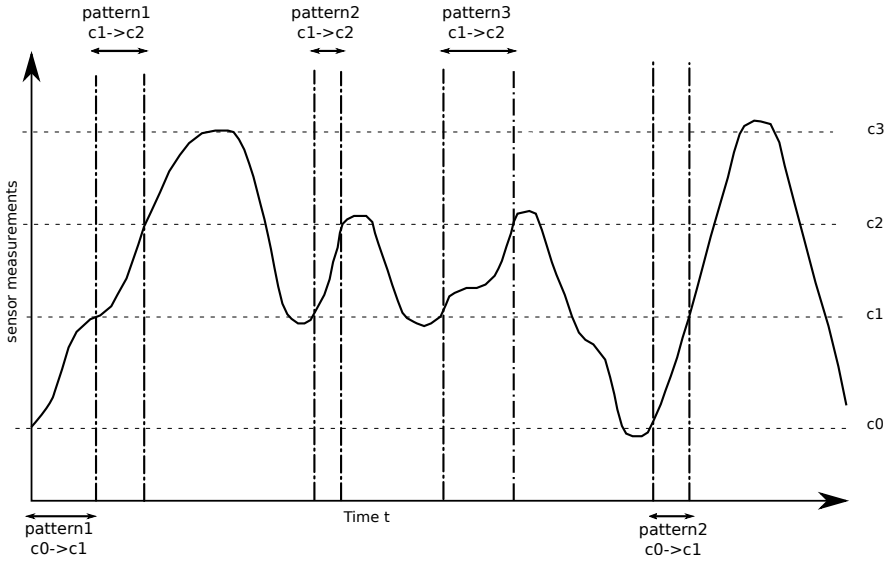


Figure 5.4: Time Pattern example

transition.

Note that this means that the IDS is *not* trying to find periodic patterns in the evolution of a sensor. The IDS looks for the time needed to complete a transition from one critical value to another. The absolute timing of when the said transition occurs is not explicitly taken into account by the IDS since transitions are treated independently and the order in which they occur is not an information used by the IDS. By doing so, the IDS is not limited to variables that exhibit periodic patterns and that must run the patterns in specific amounts of time. Imagine the example in Figure 5.4 showing the evolution of a sensor variable. The variable has four critical values $c0, c1, c2, c3$. What is visualized here is that when a critical value is reached, changes in the behavior of the variable can occur because of an action in the underlying physical process. For example, the variable increases until it reaches a critical value, which then leads to a modification of the state of an actuator related to this sensor causing the variable to start decreasing.

There are three transitions in increasing order, $c0 \rightarrow c1, c1 \rightarrow c2, c2 \rightarrow c3$ and three in decreasing order for a total of six transitions. Looking at the three transitions from $c1$ to $c2$, we can observe that they have different durations. Similarly, the two occurrences of the transition $c0 \rightarrow c1$ indicated in the figure take different amounts of time. This is the information that the IDS monitors explicitly, and not the time when a transition has started. Obviously, the absolute starting time of a transition depends on the transitions occurring before it.

In general, we observe for a process variable multiple occurrences (t_i, u_i) of a transition from critical value c_i^s to critical value c_j^s with $i \neq j$. The transition times of those occurrences are not necessarily constant, as visible in Figure 5.2a, and might vary randomly due to noise or depending on internal or external factors. During the detection phase (see below), the IDS will see a transition $c_i^s \rightarrow c_j^s$ occur with time t' and update step u' and will have to decide whether (t', u') is sufficiently similar to the occurrences (t_i, u_i) observed during the learning phase.

Unfortunately, this task is complicated by the presence of outliers. The latter can appear during the detection phase as well as during the learning phase. Most of those outliers are harmless in the sense that they are caused by noise and should therefore be accepted as legitimate. However, others are anomalous and should be ignored during the learning phase (e.g., because they are caused by unique incidents) and raise an alarm during the detection phase. The challenge is to find a distance function and threshold θ that allows us to tell apart anomalous outliers from legitimate data.

To overcome this challenge, we use the unsupervised anomaly detection algorithm called Local Outlier Factor (LOF) [Bre+00]. LOF is an algorithm that assigns to each data point a score representing the local density deviation with respect to its neighbors. Intuitively, an outlier (or anomaly) is a data point with a lower density than its neighborhood, such that outliers caused by noise are close to a cluster (but are not part of it) while anomalous outliers are far from all clusters. The idea of using LOF is to distinguish between data points that are part of a cluster and the outliers. By computing the density score, we can estimate a threshold for the score that determines whether a new data point is an outlier or not.

LOF computes for every point a ratio of its density compared to its k nearest neighbors. It uses a distance called the *reachability-distance*. Let D be a domain $\mathbb{R} \times \mathbb{R}$ and let $d : D \times D \rightarrow \mathbb{R}$ be the Euclidean distance between two points[1]. For $P \in D$ with the coordinates (p_1, p_2) and $Q \in D$ with the coordinates (q_1, q_2), $d(P, Q)$ is defined as:

$$d(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

Let $N_k(P)$ be the set of the k nearest neighbors to P according to d. Let a distance function $d_k : D \rightarrow \mathbb{R}$ for a point P be defined as

$$d_k(P) = max(d(P, Q)), \quad \forall Q \in N_k(P)$$

$d_k(P)$ is the Euclidean distance of the kth nearest neighbors of P, meaning the distance to the point included in the k nearest neighbors but that is the furthest from P.

[1] We only explain here the definition in a two-dimensional space since this is what we use in our IDS. The general definition can be found in [Bre+00]

With d and d_k, let the reachability-distance $\text{rd} : D \times D \longrightarrow \mathbb{R}$ for two points P and Q be defined as

$$\text{rd}(P, Q) = max(d_k(Q), d(P, Q))$$

The algorithm defines the local reachability density lrd as

$$\text{lrd}(P) = \left(\frac{\sum_{Q \in N_k(P)} \text{rd}(P, Q)}{|N_k(P)|} \right)^{-1}$$

The local reachability density is used to compute the Local Outlier Factor of a point P[Bre+00]:

$$LOF(P) = \frac{\sum_{Q \in N_k(P)} \frac{\text{lrd}(Q)}{\text{lrd}(P)}}{|N_k(P)|}$$

The LOF of a point is the average of the ratio of the local reachability density of its neighbours and its own. The greater the value, the more its density varies significantly from its k nearest neighbours. An outlier close to a cluster will have a lower LOF than an outlier far from every cluster. We compute the LOF of every data point and we split the resulting LOF values into two classes using the Jenks Natural Break Classification Method [Jen67]. This method is a 1-dimensional classifying method that aims at minimizing the intra-class variance and maximizing the inter-class variance. This algorithm requires to choose the number of class beforehand and in this case, we select 2 classes for non-outliers and outliers. It is an iterative process that works as follows given a set of datapoints $X = [x_0, x_1, \ldots, x_n]$ with mean $\mu(X)$ and such that $x_i \leq x_j$ for $i \neq j$.

1. It computes the Sum of Squared Deviations for array mean, $\text{SDAM}(X) = \sum_{i=0}^{n} (x_i - \mu(X))^2$
2. Create 2-classes assignements
   $C = [(x_0), (x_1, \ldots, x_n)], \ldots, [(x_0, \ldots, x_{n-1}), (x_n)]$
3. For each possible assignement in C:
   - Compute the Sum of Squared Deviation for class mean, SDAMC = $\text{SDAM}(C_{i0}) + \text{SDAM}(C_{i1})$ where C_{i0} is the set of points belonging to the first class in the i^{th} assignement. C_{i1} is the second class.
   - From all SDAMC computed takes the one with the minimum value which indicates the class assignement with the lowest in-class variance.

Using this approach, we end up with each data point assigned to either the outlier class or the non-outlier class. By selecting the LOF score of the most

deviating data point of the non-outlier data class, we can select a threshold θ such that any point with a LOF higher than θ is considered as an outlier. A datapoint x is considered as an outlier if $LOF(x) > \theta$.

To summarize, the time pattern extraction yields for each variable p and each of its transitions $x \rightarrow y$ with $x \neq y$ being critical values of p the list of observed occurrences $O^p_{x \rightarrow y} = [(t_0, u_0), (t_1, u_1), \ldots, (t_n, u_n)]$, and the LOF threshold $\theta^p_{x \rightarrow y}$. If $O^p_{x \rightarrow y} = \emptyset$, it means that the transition does not occur. For example, in our three-tank scenario in Section 5.1, a transition of Tank 3's level from 0 to 60 without passing through 20 is not possible. An attacker can overwrite a sensor value to trigger an illegal transition, but by learning which transitions should occur the IDS detects such an event.

We also define the *stillness time* for each critical value x of a discrete variable. It gives the time a variable stays in a critical value before moving to a non-critical or another critical value. Stillness times are particularly interesting for discrete variables representing actuator states, which often stay constant over extended periods, for example when the ICS waits for a continuous variable, like the tank level, to reach a certain threshold value. We also apply the clustering and LOF algorithm to them, however without the mean update step u component. The result is again a list of observed occurrences $O^p_{x,still}$ and a LOF threshold $\theta^p_{x,still}$.

5.3.4 Detection Phase

During the detection phase, the IDS follows the values of all process variables. When it detects the occurrence of a transition $x \rightarrow y$ (as defined in the previous section) for a process variable p, it first checks whether $O^p_{x \rightarrow y}$ is empty. If it is empty, the transition is not allowed and an alert is raised. If it is not empty, it computes the new occurence of the transition by computing the transition time and mean update step (t', u') and the $LOF(t', u')$ as if the new data point (t', u') was part of the data points observed during the learning phase. If the LOF is greater than the threshold $\theta^p_{x \rightarrow y}$, the occurrence is regarded as an anomalous outlier and an alert is raised. For discrete variables, the IDS also (a) checks whether the current value is a critical value, and (b) monitors the stillness times and compares their LOFs to the respective thresholds.

The IDS does not always have to wait until the end of a transition to perform a detection. For each process variable, the IDS continuously keeps track of the time passed since the beginning of a transition from a critical value and triggers the above detection process if it exceeds the maximum duration observed in the training phase for any transition from that critical value. Mathematically, let s be a continuous sensor variable with critical values $C_s = [c^s_0, \ldots, c^s_n]$. For a transition occuring in the learning phase $c^s_i \rightarrow c^s_j$,

let a function $ttime : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ give the maximum transition time for the transition:

$$ttime(c_i^s, c_j^s) = max(t_0, t_1, \ldots, t_n)$$

where $t_0, t_1, \ldots, t_n$ are the different amounts of time taken to complete the transition, as observed during the learning phase.

Let $V_{c_i^s} = [c_j^s]$ be the set of critical values c_j^s such that there is a transition $c_i^s \rightarrow c_j^s$. For the critical value c_i^s, we define the maximum (learned) duration for any transition from the critical value c_i^s as

$$maxT_{c_i^s} = max(ttime(c_i^s, c_j^s)) \quad \forall c_j^s \in V_{c_i^s}$$

Similarly, the stillness times of actuator variables are continuously checked. How aggressively the observed duration should be checked against $maxT_{c_i^s}$ depends on the type of physical process to be monitored. In a sensitive physical process with strict time constraints, $maxT_{c_i^s}$ can be seen as a strict limit. However, in practice, many physical processes have relatively soft time constraints and being too strict would trigger a series of false alerts (one per time step) in the case of noise or incomplete training data. For such processes, our IDS allows to use a relaxed maximum duration $\delta \cdot maxT_{c_i^s}$ instead, with $\delta \geq 1$ being a configuration parameter set by the operator and with default value 1.

The above temporal analyses of transitions are the core of the detection process of our IDS. Nevertheless, it also performs other checks to detect other kinds of attacks. During the detection phase, the IDS checks for each variable p whether its current value v is within the range of values observed during the learning phase. The goal is to detect overflow/underflow attacks where values go beyond the process variable limits. Because of the noise, an exact comparison with the maximum learned value max_p (resp. minimum min_p) could lead to false alerts. We therefore check if $\frac{|max_p - v|}{max_p - min_p} \leq \epsilon$ or $\frac{|min_p - v|}{max_p - min_p} \leq \epsilon$ where ϵ is by default set to 0.01.

5.4 Evaluation

5.4.1 Methodology

We evaluate our IDS on two case studies. The object of the first case study is a process running in an ICS testbed and the second one runs in an ICS simulation. Getting access to a real SCADA system is challenging as operators are not inclined to allow researchers to perform attacks on their systems. We have chosen those two case studies for the realism and the range of the attacks that can be performed. The physical processes both come from the literature and have been used to evaluate Intrusion Detection Systems in the past. For

each case study, there are two traces: an attack-free trace that we use for learning, and a trace containing normal operations and attacks that we use to test the performance of our IDS. A data point in a trace is a one-second snapshot of the state (i.e. the values of the process variables) of the physical process. In the attack trace, each data point has to be classified as malicious or legitimate by the IDS. As usual, we denote correctly classified data points as true positives (TP) resp. true negatives (TN), and wrongly classified ones as false negative (FN) resp. false positive (FP). We also define the detection rate $DR = \frac{TP}{TP+FN}$ and the false positive rate $FPR = \frac{FP}{FP+TN}$.

The attacks performed on the processes have different goals such as damaging a component of the process, or halting operations. However, we can identify three basic types of actions:

- Actuator modification: The attacker modifies the state of an actuator, for example opening a valve that was closed.
- Actuator blocking: The attacker prevents an actuator from changing state, for example leaving a valve closed when it is supposed to be opened.
- Sensor rewriting: The attacker overwrites sensor values.

The studied datasets contain 36 attacks (SWaT process), respectively 17 attacks (simulated process). Some of the attacks use more than one of the above three types of actions. The column "Occurrence" in Tables 5.2 and 5.5 shows how often a specific type of action occurred in the attack datasets of the two processes. The occurrence of an action covers the entirety of the period during which the modifications performed by that action are effective. Note that the length of that period will depend on the attack and can be longer than one second. An action counts as detected by an IDS if the latter raised at least one alarm during the activity period of the attack the action was part of. For a detailed description of the attacks on the SWaT process refer to [AM16].

The SWaT process is a fairly complex process and we have to be careful how we count false positives in our experiments. When an attacker launches an attack on a process variable, there might be a direct effect at the time of the attack, but depending on the attack and the implicit dependency relationships existing between process variables, other effects might begin to appear way after the activity of the attacker has ended. For example, one of the attacks in the trace still has an impact on the physical process two hours after the attack stopped. The complexity of the SWaT process makes it difficult to accurately pinpoint which after-effect is linked to which attack. Therefore, we consider malicious states a bit differently than Feng *et al.* [Fen+19] who also use this process. In their work, the states of the physical process inside the time window of an attack, i.e. the time between the start and the end of

the attacker's actions, are labelled as malicious. We also take this approach but we add the possibility that transitions outside of an attack window can be impacted as well. The reasoning is the following: the SWAT process operates like a pipeline where events occur in a specific order. For example, when a tank is filled, a valve will be opened. An attack on a process variable can impact the transition of an other one that depends on the targeted variable. As the impact can happen after the attack, we label the states containing such impacted transitions as malicious as there are a consequence of the attack. To identify these transitions, we use an heuristic which consists of looking at all the transitions starting before an attack period and finishing at the end of the period. Among those transitions, we only keep the unique ones for each recurrent process variable as the uniqueness indicates that these transitions are a result of the attacks.

We compare our IDS to the process-aware Invariant-based IDS by Feng *et al.* [Fen+19] and to the Prediction-based IDS using auto-regression by Hadvziosmanovic *et al.* [Had+14] that we have already described in detail in Section 5.2. We have chosen those two works for the comparison because:

1. They have been specifically designed to detect process aware attacks.

2. Similar to ours, those two works follow a data-driven approach where the detection model is automatically learned from the learning trace. Other approaches, such as [AM16; AM17; Car+11], create their models from the design specification of the physical process, which means that their performance is limited by how accurately the author of the specification has described the physical process.

3. They are representatives of two existing major approaches to detect process aware attacks, namely, invariant testing and prediction. We think that it is interesting to study the respective strengths and weaknesses of each approach.

4. Finally, we also selected those works because of their visibility in the scientific community.

In the following, we will abbreviate our Time-based IDS as T-IDS, the Invariant-based IDS as I-IDS, and the Prediction-based IDS using autoregression as AR-IDS[2].

[2]The open source implementation of the autoregressive model described in [Had+14] has a high FPR when applied to our test data. In the SWaT process, the FPR was higher than 80% and in the simulated process it was close to 50%. Most of these alerts were raised because of some variables not behaving exactly the same as during the learning phase, even when no attack was performed and also because of the F-test. The F-test relies on the size of the two sample sets (residuals and prediction errors) to determine whether the variance of the prediction error

5.4.2 Case study 1: SWaT Physical Process

5.4.2.1 Description

The SWaT physical process [AM16] is a scaled down water treatment plant created for cyber-security research and implemented in a testbed with 24 sensors and 27 actuators. The testbed is designed to produce doubly filtered water. Figure 5.5 gives an overview of the process. It consists of six stages: In the first stage, raw water is moved to a tank and then passed to the second stage where its quality is assessed. In the third, fourth, and fifth stages, several undesirable materials are removed. After those stages, the water quality is assessed again and depending on the result the water is either stored or transferred back to the third stage to clean it further.

The authors of [AM16] provide two datasets. The attack-free trace contains the values of the process variables collected over for seven days. The attack trace has a duration of four days and contains 36 attacks with different goals, such as attacks that cause an overflow/underflow of tanks, attacks that damage equipment like pipes by controlling the pumps, or wasting resources by overwriting measured values to confuse the control server. The attack mechanisms can be roughly classified into three categories: modifying the state of an actuator, locking an actuator in a state, and rewriting a sensor value. Some of the attacks fall into more than one category.

5.4.2.2 Results

Table 5.1 shows the detection results obtained for I-IDS, the AR-IDS, and our T-IDS. Feng *et al.* [Fen+19] did not make all details of their solution publicly available and we could not reach them, so we tried our best to replicate their results. The first column of Table 5.1 contains the result they have reported in [Fen+19] and the second column gives the results of our re-implementation of their IDS. The differences between the two come from missing parameter values in [Fen+19]. We have set those parameters to obtain results as close as possible to the reported results. Despite those differences, we believe that our implementation is close enough to provide a basis for comparison.

---

is higher, but as the physical process will keep running, the sample size will keep increasing in such a way that the F-test will keep rejecting the null hypothesis (i.e the two variances are the same) even for really insignificant deviations. One solution to that problem would be to limit the sample size but that would add a new operational parameter to the IDS, potentially for each variable. To avoid this additional complexity, we have replaced the F-test by a 6σ test, i.e. we test after each prediction whether the prediction error X is $\mu - 3\sigma \leq X \leq \mu + 3\sigma$ where μ and σ are the mean and standard deviation of the residuals obtained during training. Note that the original assumption of the F-test is that X is normally distributed, therefore $P(\mu - 3\sigma \leq X \leq \mu + 3\sigma)$ should be approximately 99.7% without an attack.

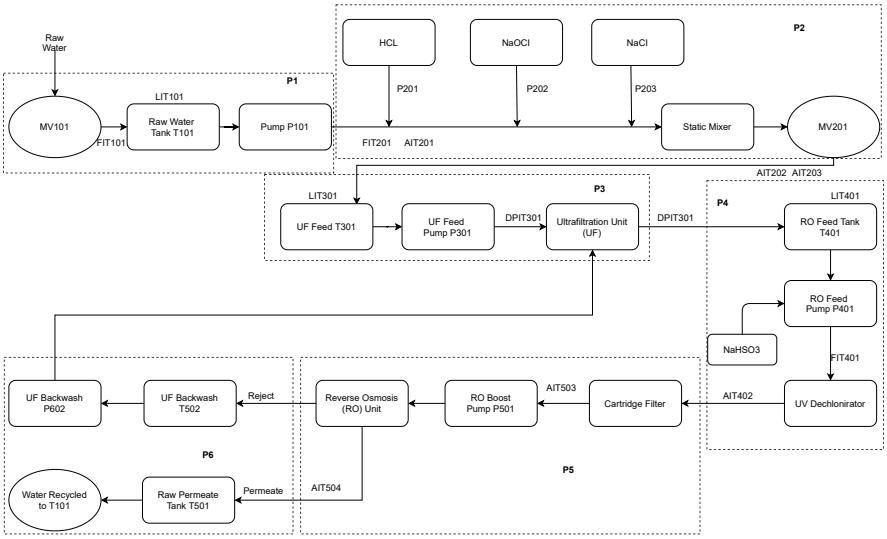


Figure 5.5: SWaT physical process (adapted from [Fen+19])

Table 5.1: Detection results for the SWaT process.

	I-IDS (results from [Fen+19])	I-IDS (reimpl.)	AR-IDS (reimpl.)	T-IDS
DR	0.7087	0.6403	0.1740	0.7266
FPR	0.0003	0.0028	0.1260	0.0095

Our T-IDS has a higher DR than I-IDS. As mentioned earlier, Invariant-based IDS can, by design, only detect an attack if it breaks an invariant rule. Among the attacks in the attack trace, the majority of them have a negative impact on the physical process that can be detected by I-IDS. However, other attacks can be seen more as attempts than successful attacks: Although they measurably influence the physical process, too, the process is able to recover in a few seconds so no invariant is violated. This shows the advantage of our T-IDS which is able to detect those attack attempts by the disturbances they cause on the time properties of the physical process. AR-IDS is only able to detect a small number of attacks and most of the ones detected are attacks where process variables go beyond their control limits. The main reason for this weak result is the nature of the attacks. AR-IDS is suited to

Table 5.2: Detection of attacks on SWaT process per type

	Occurrence	I-IDS	AR-IDS	T-IDS
Actuator modification	12	5	1	7
Actuator blocking	10	4	5	7
Sensor rewriting	25	13	14	17

detect sudden changes in process variables but many of the attacks performed on the SWaT process operate slowly and over longer periods. Moreover, some attacks consist of opening or closing valves, but given how discrete variables are treated by AR-IDS, the attacks are ignored because the discrete variables are still in a legitimate state. For the AR-IDS to detect such attacks, continuous variables must be affected as well.

Detection results per attack type can be found in Table 5.2. How well an attack is detected by the different IDS depends on its type. Overflow/underflow attacks against tanks are easily detected by all three IDS as they exceed the limits of the variables learned during the training phase. Some attacks that tamper with the measurements by overwriting the memory of the field devices. Depending on the new values, our T-IDS might detect such attacks if they cause invalid transitions. The AR-IDS detects them because they lead to drastic changes in the stream of values for a variable. I-IDS detects them only if they break an invariant, which is often the case. Finally, some attacks in the attack trace are more subtle. For example, one of them consists in increasing slightly the value given by a sensor every second. This attack is detected by our T-IDS because it impacts the transition time and the update mean step to the next critical value. It is also detected by I-IDS because the increase of the sensor value does not follow the normal distribution computed for the updates of the sensor. AR-IDS is not able to detect it because the increases are not drastic enough to cause a significant deviation.

FPR comes from how we define the threshold to consider outliers. The threshold is selected as the LOF value of the "most outlier" among the non-outliers in the learning trace. In the detection phase, because of the noise, the threshold is exceeded sometime (even slighty) for legitimate transition which cause false alert. We decide to detect more attack by sacrificing a bit the FPR. The noise is also the reason for the false alert in the AR-IDS.

For I-IDS, the FPR is very close to zero, which is expected as the mining method used in only keeps invariants with a high confidence level. The few false positives are due to coincidental invariants violated in a legitimate state.

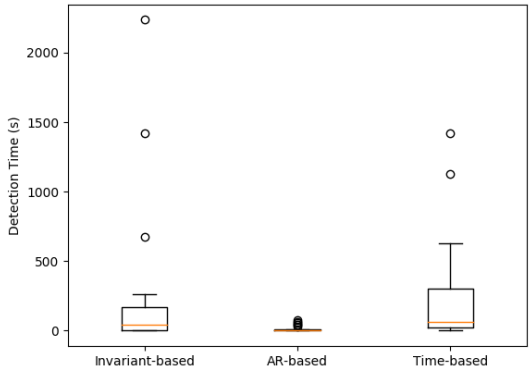


Figure 5.6: Time to detection: SWaT process

5.4.2.3 Time to Detection

Figure 5.6 presents the time needed by each IDS to detect attacks. It is computed as the difference between the timestamp of when the first alert concerning an attack was raised and the start of the attack. The average times to detection are 307s (I-IDS), 10s (AR-IDS), and 237s (T-IDS). AR-IDS' low and nearly constant time to detection reflects the fact that it performs a prediction at each update. In contrast, T-IDS has to wait until a critical value is reached or the maximum transition time of the previous critical value is exceeded. I-IDS waits until an invariant is broken. How long this takes depends on the current system state and the type of attack, hence the variability visible in the figure.

5.4.2.4 Impact of Parameter δ

Most of the alerts raised by the T-IDS for the SWaT process are the result of abnormally long stillness times of the discrete variables. For example, the stillness times of the two valves MV1 and MV2 are responsible for more than 30% of the alerts. As explained in Section 5.3, the parameter δ allows to control the detection process performed when the current stillness time exceeds the expected one. Figure 5.7 shows the impact of different values of δ on the performance of the T-IDS.

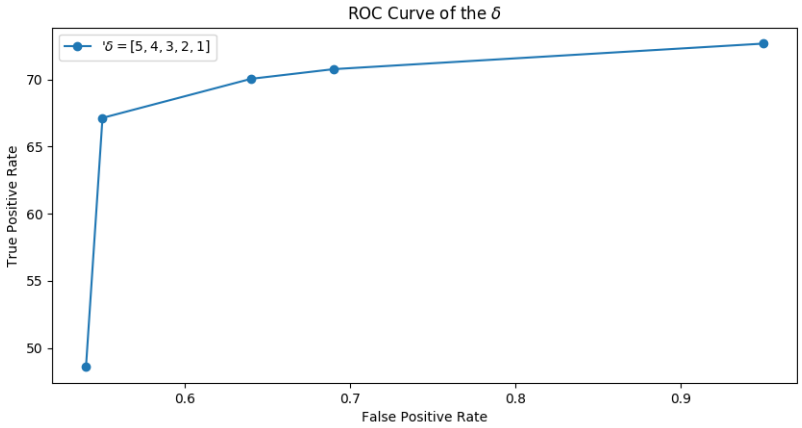


Figure 5.7: Impact of δ on Receiver Operating Caracteristic (ROC)

5.4.3 Case study 2: Simulated Physical Process

5.4.3.1 Description

In our second experiment, we evaluate the IDS on a simulated physical process. Our starting point is the physical process described in [Kou+16] and depicted in Figure 5.8. Its goal is to release a product by mixing several other products. The process is composed of two stages. In the first stage, 20 units of products are released from *Supp. 1* and *Supp. 2*, in that order. *Motor 1* mixes the products and the result is transferred to *Silo 1*. Then the second stage begins: 20 units of the product stored in *Supp. 3* are carried by a wagon to *Silo 2*. The content of *Silo 1* is added and *Motor 2* mixes them. The final product is then stored in *Tank 2*. The physical process has 23 process variables:

- One for each tank/silo, representing the level of product (7 in total).
- One for each valve representing whether the valve is open or not (7 in total).
- One for each motor representing whether it is running or not (2 in total).
- One for each motor representing the rotating speed of the blades (2 in total)
- Two indicating that the wagon has reached the left resp. right end position.
- One representing whether the wagon is moving.

Table 5.3: Simulated flushing and filling times

Container name	Flushing times	Filling times
Tank 1	20s, 4s, 8s	8s, 20s
Silo 1	10s, 8s	–
Silo 2	12s, 6s	–
Wagon	10s, 4s	4s, 10s
Tank 2	12s, 6s	–

- One representing the level of product in the wagon.
- One representing whether the trap door of the wagon is opened.

We have modified the process in order to mimic how the behaviour of a physical process could vary in reality, e.g. depending on the consistency of the raw materials. We vary dynamically how fast products are transferred from one container to another. Table 5.3 gives the flushing times used for the silos and tanks for the physical process, that is, the times to completely empty them, and the filling times, i.e. the times to completely fill them from the supply containers. Note that the supply containers have an infinite capacity and therefore no flushing time. For each flush or fill operation, the time to complete the operation is picked by the simulator from the table in a round-robin fashion. Furthermore, we increase the speed of *Motor 1* in two steps (from 0 to 20 to 40) when the motor is switched on.

We have implemented the scenario in a simulator written in python. It simulates both the physical process and the ICS components (MTU and PLCs). The simulation of the process is done with `simpy` [21c]. PLC and MTU use `pymodbus` [21b] to exchange Modbus/TCP messages on the local interface of the computer. The PLCs and the physical process run in separate python processes and interact through files. The physical process writes the current values of the process variables into a file read by the PLCs. In the opposite direction, the PLCs write to files in order to trigger changes in the physical process, e.g. to open a valve.

We generate two traces with the simulator. One attack-free trace of one hour and a trace of 30 minutes where 17 attacks are performed. The attacks are stealthy attacks, i.e. they do not try to damage the physical process in an obvious way, for example by overflowing a tank. Instead, they disturb its progress while trying to keep the process variables in a valid state. The attacker attempts to slow down the whole physical process by closing valves. The first attack starts ten minutes after the beginning of the simulation by sending commands to the PLCs to close one of the valves for one minute. After the attack, the attacker waits for a period of 5 minutes and then repeats the attack with a different valve and so on. Note that the attacker runs the

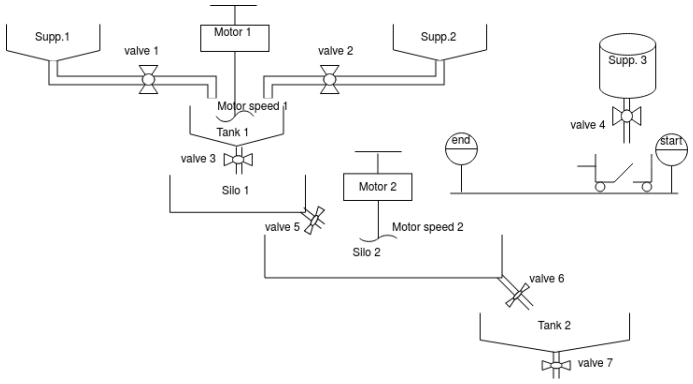


Figure 5.8: The simulated process

Table 5.4: Detection results for the simulated process

	I-IDS (reimpl.)	AR-IDS (reimpl.)	T-IDS
DR	0.52	0.2582	0.7670
FPR	0.000	0.0416	0.0320

attacks blindly, i.e., without knowing the current state of the process. Therefore, the impact of an attack and how easily it is detected by the three IDS will depend on the state of the process in the moment when the attack arrives. Other attacks consist of overflow/underflow attacks, and manipulating sensor readings such that the transitions are completed in a valid time even though updates are not done in the expected way.

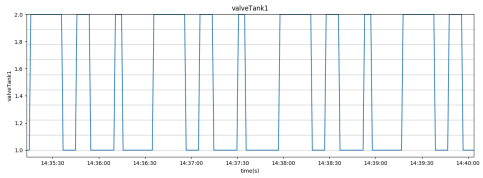
5.4.3.2 Results

Table 5.4 shows the detection results. Detection results per attack type can be found in Table 5.5.

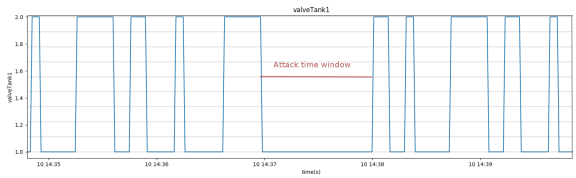
We observe that I-IDS has again a very low *FPR* because its invariant generation procedure captures very well the normal, attack-free behavior of the simulated system. However, its detection rate is low due to those attacks that were executed in such a way that no invariant was violated. An example of such an attack is when the attacker keeps *Valve 3* closed when *Tank 1* has reached 40 units of products. In that case, the normal behavior of the physical process is for *Valve 3* to open and release the product in *Tank 1*. However the state of having 40 units of product in *Tank 1* and the *Valve 3* closed is normal since it is the one occurring just before *Valve 3* is opened by the ICS, therefore no invariant is violated when the attacker keeps that state for one minute. It does not break the process but it disrupts its progress seriously

Table 5.5: Detection of attacks on simulated process per type

	Occurrence	I-IDS	AR-IDS	T-IDS
Actuator modification	1	1	1	1
Actuator blocking	5	3	5	5
Sensor rewriting	13	13	5	13



(a) Valve 3 timeseries (normal)



(b) Valve 3 timeseries (attacked)

Figure 5.9: Attack impact on Valve 3 (Values on y-axis: 2="Open", 1="Closed")

which is undesirable. T-IDS is able to detect the attack as the times of stillness of the process variable of *Valve 3* are impacted. Figure 5.9 shows how the state of *Valve 3* varies during the normal behavior of the physical process and during an attack. Note, however, that in the case where the attacker closes *Valve 3* when *Tank 1* is getting *empty*, both I-IDS and T-IDS are able to detect the attack: the former because the attack breaks an invariant, and the latter because it impacts the stillness time of *Valve 3* in its "Open" state. Just like for the previous process, the detection rate of AR-IDS is quite low because of how the attacks target the process.

5.4.3.3 Time to Detection

The detection times are shown in Figure 5.10. The average times to detection are 2.5s (I-IDS), 2.9s (AR-IDS), and 5.1s (T-IDS). For most of the attacks, T-IDS is faster than the other two IDS. However, for a few attacks, it needs much longer. This is caused by the way how the attacks are performed. As mentioned earlier, the attacks are run without knowledge of the current state of the process. For example, an attack might try to close a valve even though the valve is already closed. In such a situation, T-IDS has to wait the maximum

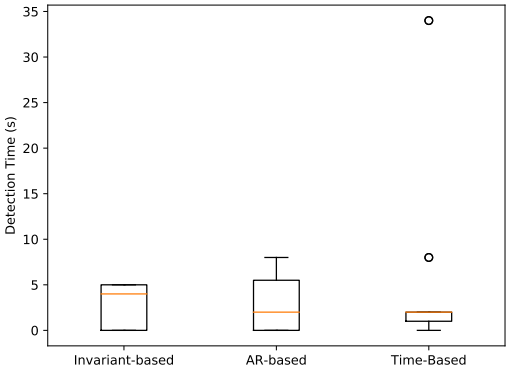


Figure 5.10: Time to detection: Simulated process

stillness time of the "Closed" state of the valve before it can raise an alert. In that regard, I-IDS is less susceptible as it captures more explicitly the dependency of that process variable with a large set of invariants that are more likely to be violated early in the case of an anomaly.

5.4.3.4 Impact of Parameter δ

For the simulated process, most of the alerts come from sensor transitions, so the value of δ has less impact compared to what was observed with the SWaT process as it can be seen in Figure 5.11. For values $2, 3, 4, 5$, the performance of the T-IDS is quite similar and not very good. This can be explained by the fact that blocking attacks are missed because the T-IDS is waiting too long to raise an alert. Therefore, by decreasing δ, the TPR increases. To conclude, the default value of 1 seems to be a good trade-off for both processes. It should be noted that the number of data points for this process is smaller than in the SWaT process, so missing a malicious point has a higher impact on the computation of the TPR (same for FPR).

5.4.4 Execution Time

All three tested IDS are able to process the data faster than real-time in the detection phase. For example, they need less than 25 seconds to classify one hour of data of the simulated process. For the learning phase, T-IDS and AR-IDS need around 30 minutes to process the entire training data, while I-IDS needs several hours due to the complexity of its data mining process.

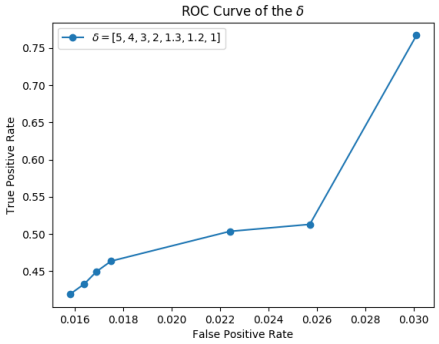


Figure 5.11: Impact of δ on Receiver Operating Characteristics (ROC) for the simulated process

5.5 Limitations

Our Time-based IDS focuses on recurrent variables with critical values. This means that if an attack is done on non-recurrent variables, it might not be detected. However, given all the dependencies that exist between variables in physical processes, an attack against one process variable has a high chance to impact others as well, meaning that the IDS might not necessarily pinpoint which process variable was targeted by an attack, but it can still detect the incident. This is what we observed in the SWaT process for some attacks.

Another limitation is the time to detection. In order to detect an attack, our IDS has to wait until a transition happens or until the longest possible transition time or stillness time for the last reached critical value is exceeded. For this reason, the IDS is not able to identify all the malicious snapshots of the physical process in the second case study, and the time it will take for the IDS to detect an attack will depend on the timing of the attacker. Imagine a situation where a valve has just been opened. If the attacker immediately closes the valve, the IDS notices the transition from critical state "Open" to "Closed". However, if the valve was already closed and the attacker prevents it from opening, the IDS has to wait the maximum stillness time of the valve's "Closed" state.

Finally, our IDS does not, in its current form, analyze exhaustively the correlation between variables. This is a strength of the Invariant-based approach [Fen+19]. To a certain degree, some level of correlation is implicitly taken into account as the dependencies between the variables are used to identify the critical values. Nevertheless, in view of the low detection rate of the Invariant-based approach in the second test case, we observe that our IDS does remarkably well.

5.6 Conclusion

Intrusion Detection Systems (IDS) that have been designed for the protection of general-purpose ICT networks can also be used to protect industrial control systems (ICS). However, they do not understand the physical process monitored by the ICS. This lack of process awareness allows attackers with knowledge of the process semantics to perform attacks harming the process itself. *Process-aware* IDS are designed to detect such attacks.

In this chapter, we looked at learning-based process-aware IDS, i.e., process-aware IDS that learn the expected behavior of the ICS and the underlying physical process from past observations. Existing approaches proposed in the literature based on the learning of invariants tend to result in very few false positives for process-oriented attacks. However, some types of attacks, e.g. attacks that aim at slowing down the physical process, evade detection because they are designed to not violate invariants.

We proposed a Time-based IDS that is able to learn the temporal properties of the physical process from a series of snapshots of the process variables. It is fast and only depends on a few configuration parameters. We have evaluated our solution on data traces from a real SCADA testbed and a simulated ICS that contain various types of attacks. Comparisons with an Invariant-based IDS and a Prediction-based IDS taken from the literature show that our IDS achieves a significantly higher detection rate.

Conclusion 6

Throughout the entirety of this thesis, we have emphasized the importance of securing ICS systems and the challenges that come with such a task. We have proposed solutions to address some of these challenges but it is important to take a step back and think about what challenges will those solutions introduce. Moreover, because of the rise in attacks against ICS, operators and manufacturers are starting to see that the current technologies, such as the protocols, are too vulnerable and since those systems are becoming targets of interest, this is a huge problem. In this last chapter, we reflect and analyze the work that has been done in this thesis. As a reminder, we listed three questions that represented the directions taken to achieve the goal of improving the security of ICS systems. These questions were:

1. Traditional IT networks and ICS networks are different. Can the differences between the two be an obstacle when using Flow-based IDS from IT in OT?

2. What defense mechanisms can a Software-Defined Networking enabled ICS use to defend itself against network oriented attacks?

3. Assuming that an attacker can tamper with the physical process through the network, how to detect their actions or their consequences?

.

This last chapter is concluded by a discussion on the perspective of ICS systems. More particularly, we take interest in the evolution of those systems with the arrival of the Internet of Things and the so-called Industry 4.0.

6.1 Reflexion and Analysis

The goal of this thesis was to design solutions aiming at improving the security of ICS systems. We first focused on the detection of intrusions in the cyber-part of ICS then we moved to the detection of process-oriented attacks.

In the context of IT networks, researchers have proposed different Anomaly-based IDS using flow-based detection techniques. The first question in this thesis was related to how the differences between IT and OT

could impact the use of those Flow-based IDS. As shown in Chapter 3, when applying such techniques to ICS network, some of them will not work as intended because the assumptions made during their design are not correct for ICS network. A typical exemple is the Entropy-based IDS which assumes that communications are varied in term of traffic profile.

In the IDS presented in Chapter 4, we show how an SDN enabled ICS can improve the Flow-Whitelisting technique but also mitigate eavesdropping. Those are two examples of mechanisms that a SDN enabled ICS can use against some network oriented attacks which answer the second question. This is just one way of how SDN can be used, others have been presented in that chapter, but the idea of benifiting of the flexibility provided by SDN is the same. It allows to adapt to new events by dynamically managing the network. The appearances or disappearances of flows are examples of events but other kind of events can be defined depending on what part of the system serves as the basis for the detection of unexpected activities. With the centralized view of the network and the southbound interface, the controller can collect statistics from the network devices like the number of packets seen or the bandwidth which are information that can help the detection[Sil+16]. There is however an important matter that must addressed. In our solutions, we assumed that SDN was a technology available in the ICS system but to our knowledge, in reality, it is not the case. However, we strongly believe that SDN will be part of such systems in the future because of all of its benefits. How to deploy SDN in existing ICS is out of the scope of this thesis but there have been works in the research community on how to incrementally deploy SDN to existing networks[Hon+16; SSH17; Lev+14]. A complete and direct transition from a traditional network to a full SDN network would be a management nightmare and financially challenging, so a possible alternative is to use a hybrid network where traditional nodes and SDN nodes operate together to provide the advantages of both network paradigms. There are however challenges that come with the coexistence of those two paradigms, like which node to upgrade to an SDN node or how the controller can get a global view of the network since it can not communicate with the traditional switches. More concerns need to be addressed but as it can be seen, applying SDN to an existing legacy network is not an easy task but the benefits are worthy enough, and not only in term of security, to consider it.

Despite the use of IDS to monitor the network, there is always a possibility, even if it is small, that an attacker goes unnoticed. This is the reason why the last question of this thesis is on how to detect an attacker who is able to disrupt the physical process through the network while bypassing the network IDS. Similarly to detection of network attack, the main trend for detecting these kind of attacks is Anomaly-based detection. The idea is to detect anomalies not at the network level but at the variable defining the

process. The assumption is that these data can be trusted meaning that they reflect what is currently happening in the physical world. In that case, the consequences of the action of an attacker can be detected. There are several proposed solution for the detection process-oriented attacks but most of them do not take into account the temporal properties of the physical process but we saw how those properties can be used in the detection of process-oriented attack. Compared to the Two-Level IDS however, the Time-based IDS is not dynamic meaning that if the semantic of the process semantics were to change or if some transitions were missed during the training phase, the Time-based IDS would need to be retrained otherwise it would trigger too many false positives.

6.2 The Future of Industrial Control Systems

Over the years and with the increasing number of attacks, ICS operators, as well as vendors, have started to understand that keeping things as they are is highly undesirable. The mindset of *if ain't broke, don't fix it* is not applicable anymore as ICS vulnerabilities cause them to be targeted — one could even say that this mindset is paradoxical. One of the main reasons that prevents the update of such systems is the constraint of availability that they need to satisfy. However, while this constraint has to be satifisfied at any time, using obsolete and insecure technologies introduces the risk of violating it. For example, by sending a simple Modbus/TCP command to a PLC, it is possible to create a Denial of Service. As the risk cannot be ignored anymore, there have been works in the industry to design new protocols with security in mind. The Modbus Organization, which designed Modbus/TCP, released in 2018 the Modbus/TCP Secure protocol [Org18], an improvement of Modbus/TCP relying on Transport Layer Security (TLS) to provide more security. TLS is a protocol widely used in the Internet providing data confidentiality, data integrity, and parties authentication.

The security aspects of ICS are not the only thing that is expected to change. As society and its technology evolve, the industry will change as well. There have been three notable industrial revolutions in the history of humankind. The first one was the introduction of mechanization via steam power, the second one was the mass production using electrical power, and the third was the introduction of computer systems and automation. For some years now, politics and industrials have entered a fourth revolution with more digitalization coming into the industrial sector. This revolution is called Industry 4.0. It includes many emergent technologies and has for goal to improve the management and the performance of manufacturing processes by using smart machines. Industry 4.0 relies heavily on smart technologies and aims to not only optimize the manufacturing processes but also

to ease the maintenance by predicting when problems are about to occur before they happen. There is not yet a precise definition of Industry 4.0 by the research and engineering community but there are nine concepts that are frequently mentioned when referring to the fourth industrial revolution [Tay+18; Imm20]:

Industrial Internet of Things (IIoT): IIoT is the use of IoT devices in an industrial context. The devices are small sensors with some level of computing power to monitor and collect data in real-time. They can be used for the processes in a factory but also to get the real-time status of the equipment and its current performance.

Big Data/Analytics: In order to optimize the processes, the amount of data that must be collected and analyzed is enormous, which is why the techniques developed in the Big Data field are necessary. Prediction of failure is typically an area where such analysis can be helpful.

Cloud Computing: With all the data collected, its storage and analysis become challenging. This is where cloud computing comes into action. It is a mature technology providing the capability to run resource-hungry tasks.

Cyber-Physical Systems (CPS) and Cyber Security: Cyber-Physical Systems are quite similar to SCADA and have been widely adopted in society. The main difference is the number of sensors and data managed. However, in Industry 4.0, the security aspect of the overall system is included in the design process.

Additive Manufacturing: Usually, the manufacturing process involves operations that remove part of the raw materials, like cutting or carving. This way of manufacturing is referred to as subtractive manufacturing. Additive Manufacturing is the opposite where pieces are built by adding layers of materials. An example of this approach is 3D printing.

Augmented Reality (AR): This technology consists of sending extra sensory information, visual most of the time, in addition to what is perceived in the real world. In an industrial context, AR can be useful for maintenance purposes. For example, providing details on a piece of equipment during maintenance.

Simulation/Digital Twin: A more precise view of the system, thanks to more data, leads to better models and therefore improves the realism of simulations. The end goal is to create a virtual factory similar to the real one to optimize equipment settings and improve maintenance.

System Integration Horizontal/Vertical: This concept describes a structural aspect of a manufacturing organization. An important aspect of Industry 4.0 is the transparency, inside one or several organizations. They communicate with each other and share experiences to improve the overall efficiency. This is the horizontal integration which can also be limited to a single organization with multiple facilities.

Autonomous Robots: Industry 4.0 promotes the use of machines that act on their own without necessitating much intervention from a human.

To our knowledge, there is not yet a concrete example of a manufacturer implementing *all* the above concepts, but individual concepts have been adapted. For example, Adidas 3D prints shoes for athletes and several companies have turned to Industrial IoT applications along with cloud computing [Cen18]. For an industrial facility to move to Industry 4.0, more connectivity between the different systems will be required, and given how more connectivity went hand in hand with more vulnerabilities in the past, it is important to think about what could be the consequences of moving to a more digitalized industry in terms of security of ICS.

First of all, in this new paradigm, almost everything relies on data and, consequently, the latter becomes a much more valuable resource. As explain in Section 2.3.1, ICS has traditionally emphasized availability more than confidentiality in the CIA (Confidentiality, Integrity and Availability) model because the data transmitted between field devices and the control server is usually not so sensitive that it needs to be encrypted. However, this will not be the case anymore in Industry 4.0. The data collected does not only include values about the physical processes being monitored and controlled but also details on the equipment itself which is valuable information for an attacker who is looking for sensitive equipment with known vulnerabilities. In that context, ensuring that data does not leak any important information is just as important as ensuring that it reaches the correct data sink and that it has not been tampered with.

Secondly, with the volume of data gathered, companies need solutions to manage those data and the primary solution for that is cloud computing. However, running one or several data centers has a cost that not all companies are able or willing to pay. The solution in that case is outsourcing to other companies and benefitting from their expertise. In terms of business, there is no issue with this solution but the same cannot be said for security as the data is now managed by an external party. The problem that ICS operators depend on other companies for certain tasks, like the maintenance of equipment, is not new. However, having data, one of the most important resources in Industry 4.0, being partially handled by other companies makes a huge difference. Data is directly involved in the business decisions and the

control of the physical processes, therefore ICS operators need to examine thoroughly the risks of letting someone else have those data. It also increases the necessity of ensuring data integrity.

Finally, there has been recently a new trend of attacks against ICS known as ransomware. Ransomware is a type of malware whose goal is to prevent the access to data stored on a device. The malware encrypts the data on the device and the attacker asks the victim a ransom in order to get the data back. An example of such an attack is WannaCry in 2017. In current ICS systems, such attacks are quite severe as the Colonial pipeline incident has shown [TM21]. In a context where every part of the system, from process control to maintenance and business decisions, relies on data, these attacks become more disruptive.

As of right now, Industry 4.0 is essentially a set of guidelines to what industries should strive for but such transition can not be done without thinking about the consequences that it entails. A lesson must be learned from the previous transition that interconnected ICS to other networks. More generally, when looking at the performance and security of computer systems, there is often a trade off that must be found between the two. We cannot expect that one can be increased without impacting the other and for that reason it is important to identify the underlying risks of reaching for more performance and to think how they can be mitigated.

6.3 Further Work

In this thesis, we have tackled the problem of detecting attacks against ICS at the network level and process level. The solutions that we propose have been evaluated and validated and while the results are encouraging those solutions have some limitations:

1. The Two-level IDS in Chapter 4 is limited by the Flow-Whitelisting approach which is vulnerable to spoofing packets. For a protocol over TCP, like Modbus/TCP, spoofing a packet to send a malicious command to a PLC is complex as an attacker has to guess the correct sequence number in order for the destination to accept the command. An UDP-based application protocol might implement a mechanism similar to TCP sequence numbers but a Flow-based IDS might ignore it since it is in the payload. In the current solution, the only mechanism against spoofed packets is their source of origin. Since they are coming from another host than the one advertised by the IP address, the location of the packet in the network is a good indication of whether or not it is spoofed. Whitelists are switch specific and can therefore detect packets with unexpected source addresses, but if the attacker is close to the

spoofed host, it has less chance of being detected. What could be done is to not only use the source location of a packet to detect spoofing attacks but also to look at the time when the packet is sent. The periodic nature of the communication between hosts in ICS can serve as a basis to detect spoofed packets with the wrong timing [LNA17].

2. The Time-based IDS in Chapter 5 does not consider process variables that are not recurrent. If these variables impact recurrent variables, then attacks targeting non-recurrent variables can be detected, but otherwise not. Such variables do not follow an obvious pattern when there are examined on their own, therefore it is difficult to model their behavior and perform anomaly detection. These variables however are part of the physical process, which means that there must be some kind of dependencies between then and other variables. By analyzing other variables and correlate their evolution over time with the evolution of non-recurrent variables, it can become easier to detect anomalies.

3. In its current state, the Time-based IDS checks whether the time pattern of a transition is valid, in the sense that it has been observed or not during the learning phase. As it has been shown, this detection process is able to detect attacks affecting the transitions of a process variable. However, it does not consider the case where a valid time pattern is observed at an invalid time. Imagine a process variable that has for the same transition two-time patterns $T1$ and $T2$. During the learning phase, the patterns occur alternately ($T1$,$T2$,$T1$,$T2$,...). In this very simple example, the IDS is able to learn the two time patterns, however, since it does not know about the alternation of the two, if an attacker is able to modify the variable pattern such that the sequence becomes $T1$, $T1$ or $T2$, $T2$, the IDS will not detect it. To detect such an attack, a possible solution is the use of Hidden Markov Models [Spe+09] where the time patterns are the states. The model would learn the sequence of time patterns and detect unexpected sequences.

4. The last subject to address is the trace generation. Given the lack of available datasets in ICS security, being able to generate synthetic traces that are similar to what can be found in real systems is valuable. However, datasets for IDS evaluation should also contain attacks. In the solution proposed in Chapter 3, the attacks added to the trace are not too intensive in the sense that they only lightly affect the overall behavior of the emulated system. This approach has limitations. For example, if an IDS misses the infection of a host by a malware, a good IDS should at least detect a posteriori that the host has been infected, e.g., by monitoring its (changed) network behavior. Such a scenario

can not be studied with the current generator as we do not emulate real end-hosts (e.g., PLC) and the behavior they would have in such a situation. To do this without a complete emulation of the end hosts, the tool could be extended by giving it samples of real host reactions to attacks.

5. More generally, we mostly focused on Modbus/TCP, but there are also other protocols with different characteristics, such as Distributed Network Protocol 3 (DNP3) or S7comm used by Siemens. The same thing can be said about our work on physical processes, as the ones studied by us are just a few among many others.

To conclude this thesis, there is an important question that must be asked: Can existing ICS implementing the solutions proposed by us and others be considered as fully secure? In other words, can we ensure the properties of the CIA model? Unfortunately, the answer is no. First of all, the main source of vulnerabilities of ICS are their legacy issues, and as long as those exist, they will be the target of attacks. Better security techniques have been proposed, for example lightweight cryptographic algorithms for resource-constrained devices [BLA17], but they have not yet been implemented in existing equipment. Secondly, for the reasons we have explained, these solutions mainly detect attacks without stopping them, which can be problematic in sensitive environments. Finally, attacks exploiting the human factor and relying on social engineering techniques like spear phishing require additional solutions, such as the training of employees to raise awareness. Nevertheless, security mechanism like IDS can help to detect and mitigate the exploitation of those attacks.

Bibliography

[18] *Capture files from 4SICS Geek Lounge 2015*. https://www.netresec.com/index.ashx?page=PCAP4SICS. accessed: October 13, 2018.

[20a] *CAIDA*. https://www.caida.org/data/overview/. accessed: 30-03-2020.

[20b] *POX Controller*. https://noxrepo.github.io/pox-doc/html/. accessed: 05-10-2020.

[21a] *NetEm-Network Emulator*. http://man7.org/linux/man-pages/man8/tc-netem.8.html. accessed: 05-10-2021.

[21b] *PyModbus, A full modbus protocol written in python*. https://github.com/riptideio/pymodbus. accessed: 05-10-2021.

[21c] *SimpPy, Discrete event simulation for Python*. https://simpy.readthedocs.io/en/latest/. accessed: 05-10-2021.

[30] *iPerf - The ultimate speed test tool for TCP, UDP and SCTP*. https://iperf.fr/. accessed: 2020-03-30.

[Al-+14] R. Al-Dalky, O. Abduljaleel, K. Salah, H. Otrok, and M. Al-Qutayri. "A Modbus traffic generator for evaluating the security of SCADA systems". In: *2014 9th International Symposium on Communication Systems, Networks Digital Sign (CSNDSP)*. July 2014, pp. 809–814.

[Alr+18] F. Alrimawi, L. Pasquale, D. Mehta, and B. Nuseibeh. "I've seen this before: Sharing cyber-physical incident knowledge". In: *Proceedings of the 1st International Workshop on Security Awareness from Design to Deployment*. 2018, pp. 33–40.

[AM16] S. Adepu and A. Mathur. "Using process invariants to detect cyber attacks on a water treatment system". In: *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer. 2016, pp. 91–104.

[AM17] S. Adepu and A. Mathur. "From Design to Invariants: Detecting Attacks on Cyber Physical Systems". In: *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 2017, pp. 533–540.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993, pp. 543–556. ISBN: 0-13-617549-X.

[Ant+17] S. D. Antón, D. Fraunholz, C. Lipps, F. Pohl, M. Zimmermann, and H. D. Schotten. "Two decades of SCADA exploitation: A brief history". In: *2017 IEEE Conference on Application, Information and Network Security (AINS)*. 2017, pp. 98–104. DOI: 10.1109/AINS.2017.8270432.

[APF15] L. O. Aday, C. C. Pastor, and A. F. Fernández. *Current Trends of Topology Discovery in OpenFlow-based Software Defined Networks*. Tech. rep. Sept. 2015.

[BDP10] A. Botta, A. Dainotti, and A. Pescape. "Do you trust your software-based traffic generator?" In: *IEEE Communications Magazine* 48.9 (Sept. 2010), pp. 158–165. ISSN: 0163-6804.

[BDP12] A. Botta, A. Dainotti, and A. Pescapé. "A tool for the generation of realistic network workload for emerging networking scenarios". In: *Computer Networks* 56.15 (2012), pp. 3531–3547.

[Bha99] R. Bhandari. "Survivable Networks: Algorithms for Diverse Routing". In: Springer US, 1999, p. 46.

[BLA17] W. J. Buchanan, S. Li, and R. Asif. "Lightweight cryptography methods". In: *Journal of Cyber Security Technology* 1.3-4 (2017), pp. 187–201.

[BM14] W. Braun and M. Menth. "Software-defined networking using OpenFlow: Protocols, applications and architectural design choices". In: *Future Internet* 6.2 (2014), pp. 302–336.

[Bre+00] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "LOF: Identifying Density-Based Local Outliers". In: *SIGMOD Rec.* 29.2 (May 2000), pp. 93–104. ISSN: 0163-5808. DOI: 10.1145/335191.335388.

[BSP12a] R. R. R. Barbosa, R. Sadre, and A. Pras. "A first look into SCADA network traffic". In: *2012 IEEE Network Operations and Management Symposium*. 2012, pp. 518–521.

[BSP12b] R. R. R. Barbosa, R. Sadre, and A. Pras. "Difficulties in Modeling SCADA Traffic: A Comparative Analysis". In: *Passive and Active Measurement*. 2012, pp. 126–135.

[BSP13] R. R. R. Barbosa, R. Sadre, and A. Pras. "Flow whitelisting in SCADA networks". In: *International Journal of Critical Infrastructure Protection* 6.3 (2013), pp. 150–158. ISSN: 1874-5482.

[BSP16] R. R. R. Barbosa, R. Sadre, and A. Pras. "Exploiting traffic periodicity in industrial control networks". In: *International Journal of Critical Infrastructure Protection* 13 (2016), pp. 52–62.

[Cao+04] J. Cao, W. S. Cleveland, K. Jeffay, F. D. Smith, and M. Weigle. "Stochastic models for generating synthetic HTTP source traffic". In: *IEEE INFOCOM 2004*. Vol. 3. Mar. 2004, 1546–1557 vol.3.

[Car+11] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. N. Fovino, and A. Trombetta. "A Multidimensional Critical State Analysis for Detecting Intrusions in SCADA Systems". In: *IEEE Transactions on Industrial Informatics* 7.2 (May 2011), pp. 179–186. ISSN: 1551-3203. DOI: 10.1109/TII.2010.2099234.

[CDT21] M. Conti, D. Donadel, and F. Turrin. "A Survey on Industrial Control System Testbeds and Datasets for Security Research". In: *arXiv preprint arXiv:2102.05631* (2021).

[Cen18] M. N. Center. https://news.microsoft.com/2018/07/16/ge-and-microsoft-enter-into-their-largest-partnership-to-date-accelerating-industrial-iot-adoption-for-customers/. accessed:05-10-2021. 2018.

[Che+07] S. Cheung, B. Dutertre, M. Fong, U. Lindqvist, K. Skinner, and A. Valdes. "Using Model-based Intrusion Detection for SCADA Networks". In: *Proceedings of the SCADA Security Scientific Symposium*. Miami Beach, Florida, Jan. 2007.

[Cho+15] S. Choi, Y. Chang, J.-H. Yun, and W. Kim. "Traffic-locality-based creation of flow whitelists for SCADA networks". In: *International Conference on Critical Infrastructure Protection*. Springer. 2015, pp. 87–102.

[CMW19] G. Constante, C. Moya, and J. Wang. "Semantic-Based Detection Architectures Against Monitoring-Control Attacks in Power Grids". In: *2019 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*. 2019, pp. 1–6. DOI: 10.1109/SmartGridComm.2019.8909721.

[CSP15] A. R. Chavez, W. M. S. Stout, and S. Peisert. "Techniques for the dynamic randomization of network attributes". In: *2015 International Carnahan Conference on Security Technology (ICCST)*. Sept. 2015, pp. 1–6. DOI: 10.1109/CCST.2015.7389661.

[CTA30] B. Claise, B. Trammell, and P. Aitken. *Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. STD 77. RFC Editor, 2013, accessed: 2020-03-30.

[CZK15] M. Caselli, E. Zambon, and F. Kargl. "Sequence-aware Intrusion Detection in Industrial Control Systems". In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. CPSS '15. Singapore, Republic of Singapore: ACM, 2015, pp. 13–24. ISBN: 978-1-4503-3448-8. DOI: 10.1145/2732198.2732200.

[DBK13] D. Dittrich, M. Bailey, and E. Kenneally. *Applying Ethical Principles to Information and Communication Technology Research: A Companion to the Menlo Report*. Tech. rep. U.S. Department of Homeland Security, Oct. 2013.

[Deb00] H. Debar. "An introduction to intrusion-detection systems". In: *Proceedings of Connect* 2000 (2000).

[Do+17] V. L. Do, L. Fillatre, I. Nikiforov, and P. Willett. "Security of SCADA systems against cyber-physical attacks". In: *IEEE Aerospace and Electronic Systems Magazine* 32.5 (2017), pp. 28–45. DOI: 10.1109/MAES.2017.160047.

[Don+15] X. Dong, H. Lin, R. Tan, R. K. Iyer, and Z. Kalbarczyk. "Software-Defined Networking for Smart Grid Resilience: Opportunities and Challenges". In: *Proceedings of the 1st ACM Workshop on Cyber-Physical System Security*. CPSS '15. Singapore, Republic of Singapore: ACM, 2015, pp. 61–68.

[E+02] B. E., C. J., E. A., and H. D. "Worlds in Collisions Ethernet and The Factory Floor". In: (2002).

[EL16] N. Evancich and J. Li. *Attacks on Industrial Control Systems*. Cham: Springer International Publishing, 2016, pp. 95–110. ISBN: 978-3-319-32125-7. DOI: 10.1007/978-3-319-32125-7_6.

[Fan+04] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon. "Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme". In: *Computer Networks* 46.2 (2004), pp. 253–272. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2004.03.033.

[FCR19] R. Flosbach, J. J. Chromik, and A. Remke. "Architecture and prototype implementation for process-aware intrusion detection in electrical grids". In: *2019 38th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2019, pp. 42–4209.

[Fen+19] C. Feng, V. R. Palleti, A. Mathur, and D. Chana. "A Systematic Framework to Generate Invariants for Anomaly Detection in Industrial Control Systems." In: *NDSS*. 2019.

[Fou09] O. N. Foundation. *OpenFlow Switch Specification,version 1.0.1 (WireProtocol 0x01)*. 2009.

[Fov+10] I. N. Fovino, M. Masera, M. Guglielmi, A. Carcano, and A. Trombetta. "Distributed intrusion detection system for SCADA protocols". In: *International Conference on Critical Infrastructure Protection*. Springer. 2010, pp. 95–110.

[Gao+10] W. Gao, T. Morris, B. Reaves, and D. Richey. "On SCADA control system command and response injection and intrusion detection". In: *2010 eCrime Researchers Summit*. Oct. 2010, pp. 1–9. DOI: 10.1109/ecrime.2010.5706699.

[Gar+09] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. "Anomaly-based network intrusion detection: Techniques, systems and challenges". In: *Computers & Security* 28.1 (2009), pp. 18–28. ISSN: 0167-4048.

[Gio+14] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris. "Combining OpenFlow and sFlow for an Effective and Scalable Anomaly Detection and Mitigation Mechanism on SDN Environments". In: *Computer Networks* 62 (Apr. 2014), pp. 122–136.

[GW13] N. Goldenberg and A. Wool. "Accurate modeling of Modbus/TCP for intrusion detection in SCADA systems". In: *international journal of critical infrastructure protection* 6.2 (2013), pp. 63–75.

[Had+14] D. Hadžiosmanović, R. Sommer, E. Zambon, and P. H. Hartel. "Through the eye of the PLC: semantic security monitoring for industrial processes". In: *Proceedings of the 30th Annual Computer Security Applications Conference*. 2014, pp. 126–135.

[HF+18] K. E. Hemsley, E. Fisher, et al. *History of industrial control system cyber incidents*. Tech. rep. Idaho National Lab.(INL), Idaho Falls, ID (United States), 2018.

[HGS18] A. Hamza, H. H. Gharakheili, and V. Sivaraman. "Combining MUD policies with SDN for IoT intrusion detection". In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. 2018, pp. 1–7.

[Hof+13] R. Hofstede, V. Bartoš, A. Sperotto, and A. Pras. "Towards real-time intrusion detection for NetFlow and IPFIX". In: *Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013)*. Oct. 2013, pp. 227–234.

[Hon+16] D. K. Hong, Y. Ma, S. Banerjee, and Z. M. Mao. "Incremental deployment of SDN in hybrid enterprise and ISP networks". In: *Proceedings of the Symposium on SDN Research*. 2016, pp. 1–7.

[Hoo95] M. J. de Hoon. *Parameter Estimation of Nearly non-Stationary Autoregressive Processes*. Delft University of Technology, 1995.

[ikm30] ikm@phrack. *Blind TCP/IP hijacking is still alive*. 2007, accessed: 2020-03-30.

[ILW06] V. M. Igure, S. A. Laughter, and R. D. Williams. "Security issues in SCADA networks". In: *Computers & Security* 25.7 (2006), pp. 498–506. ISSN: 0167-4048.

[Imm20] G. Immerman. *Emerging Industry 4.0 Technologies with Real-World Examples*. https://www.machinemetrics.com/blog/industry-4-0-technologies. accessed: 05-10-2021. 2020.

[Jen67] G. Jenks. "The Data Model Concept in Statistical Mapping". In: 1967.

[Kaba] G. Kabasele Ndonda. *Eavesdropping mitigation case Study*. https://github.com/gkabasele/eavesdropping_case_study. accessed:09-01-22.

[Kabb] G. Kabasele Ndonda. *Simuled SCADA physical process*. https://github.com/gkabasele/ics-waterdistribution. accessed:09-01-22.

[Kabc] G. Kabasele Ndonda. *Time-based IDS for detecting process-oriented attacks*. https://github.com/gkabasele/ProcessBasedIDS. accessed:09-01-22.

[Kabd] G. Kabasele Ndonda. *Trace Generation for Flow-based IDS evaluation*. https://github.com/gkabasele/traces_analysis. accessed:09-01-22.

[Kabe] G. Kabasele Ndonda. *Two-Level IDS for ICS/SCADA*. https://github.com/gkabasele/whitelist-IDS. accessed:09-01-22.

[Kam+02] P. Kamath, J. Heidemann, J. Bannister, and J. Touch. "Generation of high bandwidth network traffic traces". In: *Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*. Oct. 2002, pp. 401–410.

[Kan+14] D.-H. Kang, B.-K. Kim, J.-C. Na, and K.-S. Jhang. "Whitelist generation technique for industrial firewall in scada networks". In: *Frontier and Innovation in Future Computing and Communications*. Springer, 2014, pp. 525–534.

[KČD12] R. Krejčí, P. Čeleda, and J. Dobrovolný. "Traffic Measurement and Analysis of Building Automation and Control Networks". In: *Dependable Networks and Services*. Springer, 2012, pp. 62–73.

[KHT95] A. M. Khandker, P. Honeyman, and T. J. Teorey. "Performance of DCE RPC". In: *Second International Workshop on Services in Distributed and Networked Environments*. June 1995, pp. 2–10.

[Kla30] A. Klassen Fred. *Tcpreplay - Pcap editing and replaying utilities*. `https://tcpreplay.appneta.com/`. 2018, accessed: 2020-03-30.

[KM+08] M. Katagi, S. Moriai, et al. "Lightweight cryptography for the internet of things". In: *Sony Corporation* 2008 (2008), pp. 7–10.

[Kou+16] O. Koucham, S. Mocanu, G. Hiet, J.-M. Thiriet, and F. Majorczyk. "Detecting Process-Aware Attacks in Sequential Control Systems". In: *21st Nordic Conference on Secure IT Systems (NordSec 2016)*. Oulu, Finland, Nov. 2016.

[Kre+15] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. "Software-Defined Networking: A Comprehensive Survey". In: *Proceedings of the IEEE* 103.1 (Jan. 2015), pp. 14–76.

[Lan11] R. Langner. "Stuxnet: Dissecting a Cyberwarfare Weapon". In: *IEEE Security Privacy* 9.3 (2011), pp. 49–51. DOI: 10.1109/MSP.2011.67.

[Lee+17] S. Lee, J. Kim, S. Shin, P. Porras, and V. Yegneswaran. "Athena: A Framework for Scalable Anomaly Detection in Software-Defined Networks". In: *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. June 2017, pp. 249–260.

[Lev+14] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann. "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014, pp. 333–345.

[LF16] A. Lemay and J. Fernandez. "Providing SCADA Network Data Sets for Intrusion Detection Research". In: *9th USENIX Workshop on Cybersecurity Experimentation and Test (CSET'16)*. 2016.

[LHM10a] B. Lantz, B. Heller, and N. McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 19:1–19:6. ISBN: 978-1-4503-0409-2.

[LHM10b] B. Lantz, B. Heller, and N. McKeown. "A Network in a Laptop: Rapid Prototyping for Software-defined Networks". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 19:1–19:6. ISBN: 978-1-4503-0409-2.

[Lin+13] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer. "Adapting Bro into SCADA: Building a Specification-based Intrusion Detection System for the DNP3 Protocol". In: *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. CSIIRW '13. ACM, 2013, 5:1–5:4.

[Lin+18] H. Lin, A. Slagell, Z. T. Kalbarczyk, P. W. Sauer, and R. K. Iyer. "Runtime Semantic Security Analysis to Detect and Mitigate Control-Related Attacks in Power Grids". In: *IEEE Transactions on Smart Grid* 9.1 (2018), pp. 163–178. DOI: 10.1109/TSG.2016.2547742.

[Liu+11] H. Liu, Y. Sun, V. C. Valgenti, and M. S. Kim. "TrustGuard: A flow-level reputation-based DDoS defense system". In: *2011 IEEE Consumer Communications and Networking Conference (CCNC)*. Jan. 2011, pp. 287–291.

[LKS05] A. Lazarevic, V. Kumar, and J. Srivastava. "Intrusion detection: A survey". In: *Managing Cyber Threats*. Springer, 2005, pp. 19–78.

[LN19] C.-Y. Lin and S. Nadjm-Tehrani. "Timing patterns and correlations in spontaneous SCADA traffic for anomaly detection". In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 2019, pp. 73–88.

[LNA17] C.-Y. Lin, S. Nadjm-Tehrani, and M. Asplund. "Timing-based anomaly detection in SCADA networks". In: *International Conference on Critical Information Infrastructures Security*. Springer. 2017, pp. 48–59.

[LRF16] A. Lemay, J. Rochon, and J. M. Fernandez. "A Practical Flow White List Approach for SCADA Systems". In: *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016*. ICS-CSR '16. Belfast, United Kingdom: BCS Learning & Development Ltd., 2016, pp. 1–4. ISBN: 978-1-78017-3573.

[Mak+12] A. Makke, O. Salem, M. Assaad, H. Moungla, and A. Mehaoua. "Flooding Attacks Detection in Backbone Traffic Using Power Divergence". In: *Proceedings of the 7th ACM Workshop on Performance Monitoring and Measurement of Heterogeneous Wireless and Wired Networks*. PM2HW2N '12. ACM, 2012, pp. 15–20. ISBN: 978-1-4503-1626-2.

[McK+08] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. "OpenFlow: Enabling Innovation in Campus Networks". In: *SIGCOMM Comput. Commun. Rev.* 38.2 (Mar. 2008), pp. 69–74. ISSN: 0146-4833.

[McK+17] K. A. McKay, L. Bassham, M. S. Turan, and N. Mouha. "Report on Lightweight Cryptography". In: (2017).

[MMS19] P. Manso, J. Moura, and C. Serrão. "SDN-based intrusion detection system for early detection and mitigation of DDoS attacks". In: *Information* 10.3 (2019), p. 106.

[Mor+11] T. Morris, A. Srivastava, B. Reaves, W. Gao, K. Pavurapu, and R. Reddi. "A control system testbed to validate critical infrastructure protection concepts". In: *International Journal of Critical Infrastructure Protection* 4 (Aug. 2011), pp. 88–103.

[Mor15] T. I. Morris T. Thornton Z. "Industrial Control System Simulation and Data Logging for Intrusion Detection System Research". In: *7th Annual Southeastern Cyber Security Summit*. June 2015.

[Nag84] J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896. RFC Editor, Jan. 1984.

[NDR16] N. G. Nayak, F. Dürr, and K. Rothermel. "Time-Sensitive Software-Defined Network (TSSDN) for Real-Time Applications". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 2016, pp. 193–202. ISBN: 9781450347877. DOI: 10.1145/2997465.2997487.

[Ner18] J. Neruda. "Configuration of Remote P4 Device". In: *Excel@ Fit 2018* (2018).

[Nie+18] M. Niedermaier, J.-O. Malchow, F. Fischer, D. Marzin, D. Merli, V. Roth, and A. Von Bodisco. "You snooze, you lose: measuring PLC cycle times under attacks". In: *12th USENIX Workshop on Offensive Technologies WOOT18)*. 2018.

[NPP17] S. Nazir, S. Patel, and D. Patel. "Assessing and augmenting SCADA cyber security: A survey of techniques". In: *Computers & Security* 70 (2017), pp. 436–454.

[NS17] G. K. Ndonda and R. Sadre. "A low-delay SDN-based countermeasure to eavesdropping attacks in industrial control systems". In: *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE. 2017, pp. 1–7.

[NS18] G. K. Ndonda and R. Sadre. "A two-level intrusion detection system for industrial control system networks using P4". In: *5th International Symposium for ICS & SCADA Cyber Security Research 2018 5*. 2018, pp. 31–40.

[NS20] G. K. Ndonda and R. Sadre. "Network trace generation for flow-based IDS evaluation in control and automation systems". In: *International Journal of Critical Infrastructure Protection* 31 (2020), p. 100385. ISSN: 1874-5482. DOI: https://doi.org/10.1016/j.ijcip.2020.100385.

[Org06] M. Organization. *MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b*. http://www.modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf. accessed: 30-03-2021. 2006.

[Org15] S. I. Organization. *Repository of Industrial Security Incidents*. 2015.

[Org18] M. Organization. *MODBUS/TCP Security - Protocol Specification*. https://modbus.org/docs/MB-TCP-Security-v21_2018-07-24.pdf. accessed: 05-10-2021. 2018.

[Pli+20] D. Pliatsios, P. Sarigiannidis, T. Lagkas, and A. G. Sarigiannidis. "A survey on SCADA systems: secure protocols, incidents, threats and tactics". In: *IEEE Communications Surveys & Tutorials* 22.3 (2020), pp. 1942–1976.

[R R12] R. R. R. Barbosa and R. Sadre and A. Pras. "A first look into SCADA network traffic". In: *2012 IEEE Network Operations and Management Symposium*. Apr. 2012, pp. 518–521.

[Rin+17] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho. "Flow-Based Benchmark Data Sets for Intrusion Detection". In: *European Conference on Cyber Warfare and Security*. 2017, pp. 361–369.

[Rin+18] M. Ring, D. Schlör, D. Landes, and A. Hotho. "Flow-based Network Traffic Generation using Generative Adversarial Networks". In: *CoRR* abs/1810.07795 (2018).

[SB04] J. Sommers and P. Barford. "Self-configuring network traffic generation". In: *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*. ACM. 2004, pp. 68–81.

[SEG18] V. K. Singh, H. Ebrahem, and M. Govindarasu. "Security evaluation of two intrusion detection systems in smart grid scada environment". In: *2018 North American Power Symposium (NAPS)*. IEEE. 2018, pp. 1–6.

[SFS11] K. A. Stouffer, J. A. Falco, and K. A. Scarfone. *SP 800-82. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such As Programmable Logic Controllers (PLC)*. Tech. rep. Gaithersburg, MD, United States, 2011.

[SG13] S. Shirali-Shahreza and Y. Ganjali. "Efficient implementation of security applications in openflow controller with flexam". In: *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*. IEEE. 2013, pp. 49–54.

[She+12] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. "Making Middleboxes Someone else's Problem: Network Processing As a Cloud Service". In: *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. SIGCOMM '12. Helsinki, Finland: ACM, 2012, pp. 13–24. ISBN: 978-1-4503-1419-0.

[She04] S. J. Sheather. "Density estimation". In: *Statistical science* (2004), pp. 588–597.

[Sil+15] E. G. da Silva, L. A. D. Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville, and A. Schaeffer-Filho. "Capitalizing on SDN-based SCADA systems: An anti-eavesdropping case-study". In: *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. May 2015, pp. 165–173.

[Sil+16] E. G. da Silva, A. S. d. Silva, J. A. Wickboldt, P. Smith, L. Z. Granville, and A. Schaeffer-Filho. "A One-Class NIDS for SDN-Based SCADA Systems". In: *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*. Vol. 1. 2016, pp. 303–312. DOI: 10.1109/COMPSAC.2016.32.

[SM08] J. Slay and M. Miller. "Lessons Learned from the Maroochy Water Breach". In: *Critical Infrastructure Protection*. Ed. by E. Goetz and S. Shenoi. Boston, MA: Springer US, 2008, pp. 73–82. ISBN: 978-0-387-75462-8,

[So +10] A. So perotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. "An Ove rview of IP Flow-Based Intrusion Detection". In: *IEEE Communications Surveys & Tutorials* 12.3 (2010). security netflow, pp. 343–3 56.

[Spe+09] A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras. "Hidden Markov Model Modeling of SSH Brute-Force Attacks". In: *Integrated Management of Systems, Services, Processes and People in IT*. Springer Berlin Heidelberg, 2009, pp. 164–176. ISBN: 978-3-642-04989-7.

[SSH17] Sandhya, Y. Sinha, and K. Haribabu. "A survey: Hybrid SDN". In: *Journal of Network and Computer Applications* 100 (2017), pp. 35–55. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2017.10.003.

[Sug78] N. Sugiura. "Further analysts of the data by akaike's information criterion and the finite corrections: Further analysts of the data by akaike's". In: *Communications in Statistics-Theory and Methods* 7.1 (1978), pp. 13–26.

[SYB04] J. Sommers, V. Yegneswaran, and P. Barford. "A framework for malicious workload generation". In: *4th ACM SIGCOMM conference on Internet measurement*. ACM. 2004, pp. 82–87.

[SYB05] J. Sommers, V. Yegneswaran, and P. Barford. *Toward comprehensive traffic generation for online ids evaluation*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2005.

[SZF18] F. Sicard, É. Zamai, and J.-M. Flaus. "Critical States Distance Filter Based Approach for Detection and Blockage of Cyberattacks in Industrial Control Systems". In: *Diagnosability, Security and Safety of Hybrid Dynamic and Cyber-Physical Systems*. Ed. by M. Sayed-Mouchaweh. Cham: Springer International Publishing, 2018, pp. 117–145.

[Tay+18] S. Tay, L. Te Chuan, A. Aziati, and A. N. A. Ahmad. "An Overview of Industry 4.0: Definition, Components, and Government Initiatives". In: *Journal of Advanced Research in Dynamical and Control Systems* 10 (Dec. 2018), p. 14.

[TM21] W. Turton and K. Mehrotra. `https://www.bloomberg.com/news/articles/2021-06-04/hackers-breached-colonial-pipeline-using-compromised-password`. accessed: 05-10-2021. 2021.

[Urb+16] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg. "Limiting the impact of stealthy attacks on industrial control systems". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1092–1105.

[VC09] A. Valdes and S. Cheung. "Communication pattern anomaly detection in process control systems". In: *2009 IEEE Conference on Technologies for Homeland Security*. May 2009, pp. 22–29. DOI: 10.1109/THS.2009.5168010.

[VL14] A. Varet and N. Larrieu. "How to Generate Realistic Network Traffic?" In: *2014 IEEE 38th Annual Computer Software and Applications Conference*. July 2014, pp. 299–304.

[Wei+06] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith. "Tmix: a tool for generating realistic TCP application workloads in ns-2". In: *ACM SIGCOMM Computer Communication Review* 36.3 (2006), pp. 65–76.

[WXG15] H. Wang, L. Xu, and G. Gu. "FloodGuard: A DoS Attack Prevention Extension in Software-Defined Networks". In: *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. DSN '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 239–250. ISBN: 978-1-4799-8629-3.

[Yan+20] Z. Yang, L. He, P. Cheng, J. Chen, D. K. Yau, and L. Du. "PLC-Sleuth: Detecting and Localizing Intrusions Using Control Invariants". In: *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID2020)*. 2020, pp. 333–348.

[Yiz95] Yizong Cheng. "Mean shift, mode seeking, and clustering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 790–799. DOI: 10.1109/34.400568.

[YUH06] D. Yang, A. Usynin, and J. W. Hines. "Anomaly-based intrusion detection for SCADA systems". In: *5th intl. topical meeting on nuclear plant instrumentation, control and human machine interface technologies (npic&hmit 05)*. 2006, pp. 12–16.

[Zei20] R. Zeidler. `https://securityintelligence.com/posts/what-the-explosive-growth-in-ics-infrastructure-targeting-means-for-security-leaders/`. accessed:22-12-21. 2020.

[ZJS11] B. Zhu, A. Joseph, and S. Sastry. "A taxonomy of cyber attacks on SCADA systems". In: *2011 International conference on internet of things and 4th international conference on cyber, physical and social computing*. IEEE. 2011, pp. 380–388.

[ZR17] Z. Zheng and A. L. N. Reddy. "Safeguarding Building Automation Networks: THE-Driven Anomaly Detector Based on Traffic Analysis". In: *26th International Conference on Computer Communication and Networks (ICCCN)*. 2017, pp. 1–11.