# Knowledge needed to develop malware to infect and impact industrial control systems

van de Wouw, D.A.

*Award date:*
2013

Technische Universiteit
**Eindhoven**
University of Technology

**Department of Mathematics and Computer Science**
Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

**Supervisor**
prof.dr. Sandro Etalle (TU/e)

**Section**
Security group

**Supervisor**
Trajce Dimkov, PhD (Deloitte)

**Section**
Security and Privacy

**Date**
October 21, 2013

**Version**
1.0

# Knowledge needed to develop malware to infect and impact Industrial Control Systems

Master's Thesis

D.A. van de Wouw

Where innovation starts

Deloitte.

Chapter

# Executive Summary

Industrial Control Systems (ICSs) have been with us for more than five decades. They are used to perform a broad range of automated tasks, such as the production of food, drinks, electricity and oil. ICSs are also used to control services like traffic lights, safe railroad crossings or transport luggage on an airport conveyor belt. Some ICSs are critical to society, like a power plant: households can experience a black-out if a power plant stops working.

ICSs were initially communicating through proprietary control protocols on specialized hardware and software, isolated from the rest of the world. Later, standardized communication protocols, hardware and software were introduced and recently businesses have started to integrate ICS with regular Information Technology (IT). These changes enable new functionality but also imply that ICSs become more accessible to the outside world and IT problems, including malicious software (malware), become a threat to ICS networks.

Malware is omnipresent and malware is now able to reach ICSs. This could cause financial loss or physical damage. Developing malware that can infect and impact ICSs requires a certain amount of prior system knowledge. If the information needed to develop malware for a specific target was kept secret by ICS managers/employees, it would be harder to target that ICS using malware.

This brought up the research question: "*What system knowledge is needed for a malware developer to create malware to infect and impact an Industrial Control System?*". We divided it into two sub-questions, the first is about the knowledge needed to *infect* ICSs and the second about the knowledge needed to *impact* the security of ICSs.

We first set up an environment to represent a chemical plant which contained drums, pumps and valves. Then, we developed malware that was able to infect the plant and impact the integrity and availability by disrupting plant supervision and overflowing or emptying the drums.

A list of possible environmental changes was prepared which was reviewed and completed by ICS and malware specialists. We changed the environment according to an item on the list and determined if the change reduced or diminished the effects of the malware. System knowledge was needed if the malware was unable to infect or impact the security of the plant when this was not caused by a design decision. These findings were analyzed and together with the learned lessons the sub-questions were answered.

The outcome of the thesis was that malware developers need to acquire certain knowledge to launch a targeted attack on an ICS. If an attacker wants to impact the security of the ICS with malware, he needs to infect the ICS first. This requires knowledge about what Operating Systems (OSs) (e.g., Windows, Linux) need to be infected. One or more exploits compatible with the OS are needed to infect the targeted machines. Knowledge about the OS version is needed, depending on the vulnerabilities that the exploits target. If the target is not connected to the Internet (i.e., completely air-gapped) then an attack scenario with corresponding exploits is needed (e.g., a scenario where the malware infects USB-drives). Knowledge about firewalls and their rules will enable the attacker to develop malware that can spread and scan through networks without being blocked. The attacker should know a (unique) property of the target for the malware to detect if it has reached its target.

If the malware reaches a machine that can control the environment it will try to impact the integrity and/or availability of the system. If the malware can create a backdoor that is able to communicate with a command and control server, then it becomes possible for the attacker to analyze the environment manually. Otherwise, more knowledge would be needed during development, such as knowledge about the physical processes that are controlled by the ICS and how the processes are controlled. Knowledge about the (Supervisory Control and Data Acquisition (SCADA) or Human Machine Interface (HMI)) software used at the ICS can be used to develop extra exploits and enumerate connected Programmable Logic Controllers (PLCs) at runtime.

The main impact of the research is that it provides a list of system knowledge that is needed to develop malware to infect and impact the security of an ICS. As part of their defence in depth strategy, ICS staff should keep the information described here secret to make it harder for malware developers to launch targeted attacks.

Chapter

# Acknowledgements

While I wrote my thesis I had a lot of support from colleagues at Deloitte, family and friends. I would like to take this opportunity to thank them. First, I want to thank my supervisors Trajce Dimkov and Sandro Etalle for guiding me through the process of performing research, reviewing my thesis and providing useful insights. I would like to thank Dmitri Jarnikov for being a voting member in the assessment committee.
Then I would like to thank my colleagues from the Security and Privacy team of Deloitte for the interesting discussions and for creating a fun environment to work in. I want to thank Floris Schoenmakers and Sergio Hernando for sharing their knowledge about Industrial Control Systems. I also want to thank Thijs Bosschert for getting me up to speed with malware (development). I would like to thank Spase Stojanovski for his opinion and suggestions regarding the setup of an Industrial Control System environment. I want to thank the participants of the questionnaire for providing valuable feedback. Finally, I would like to thank my family, friends, and girlfriend for their support.

Chapter

# Contents

# Chapter 1 Introduction

Industrial Control Systems (ICSs) have been with us for more than five decades. They are used to perform a broad range of automated tasks, such as the production of food, drinks, electricity and oil. ICSs are also used to control services like traffic lights, safe railroad crossings or transport luggage on an airport conveyor belt. Some ICSs are critical to society, like a power plant: households can experience a black-out if a power plant stops working.

ICSs were initially communicating through proprietary control protocols on specialized hardware and software, isolated from the rest of the world. Later, standardized communication protocols, hardware and software were introduced and recently businesses have started to integrate ICS with regular Information Technology (IT). These changes enable new functionality such as remote supervision and presenting production information to the business side of the company. The additional functionality enables cost efficient operations or increased production. The integration also implies that ICSs become more accessible to the outside world and IT problems, including malicious software (malware), become a threat to ICS networks.

ICSs have been a target by increasingly sophisticated attacks. An attack in 2000 targeted a sewage control system in Queensland [27]. The attacker used a laptop and a radio transmitter to take control of 150 sewage pumping stations right after the system was installed. Over a three-month period the attacker released one million liters of untreated sewage into a stormwater drain from where it flowed to local waterways.
One of the most famous attacks, Stuxnet, consisted of malware that targeted an Iranian uranium enrichment facility in 2009-2010. Its purpose was to stop or at least delay Iran's nuclear program. It reprogrammed uranium enrichment centrifuges controllers to adjust their spinning speed. The result was that Stuxnet destroyed more than a thousand uranium enrichment centrifuges.

## 1.1 Scenario

To set the context of the thesis, we use a scenario. Consider a chemical plant which mixes chemicals in large drums until the right ratio is obtained. An Industrial Control System (ICS) controls this process with sensors and actuators. The whole process is supervised by engineers in the control room. The plant is also connected to the Internet so it can be supervised from home.

Consider the following two attack scenarios: In scenario one, an employee wants to supervise the plant from a remote location. He will do that from a computer or laptop that is connected to the Internet. If that machine is infected or taken over, it will create an opening for an attacker to gain control over a machine in the plant. In attack scenario two, an infected Universal Serial Bus (USB) stick with malware is dropped near the plant. If the USB stick gets inserted inside the plant then the malware can infect a plant that is otherwise isolated from the Internet (air-gapped).

Once the malware gains access to the plant it could create have different implications. Possible outcomes of a malware infected plant could be one of the following: The malware can overflow drums with chemicals which causes production losses and environmental damages. An outcome could be that the malware causes the chemicals to be mixed in the wrong ratio. A wrong ratio can have several implications; it can create production losses, possibly endanger health if the plant produces flavours or drinkable products or initiate a chemical reaction that causes a fire or explosion at the plant. Yet another outcome could be that the malware tries to make the plant uncontrollable by performing an attack on the supervisory and control system. An uncontrolled plant could cause any of the above outcomes.

## 1.2 Problem description

Malware is omnipresent and while ICS tailored malware exists, most malware is intended for regular IT infrastructures. The problem is that malware can now reach ICSs and when it does, it could cause financial loss or physical damage.

To develop malware which can infect and impact ICSs requires a certain amount of prior system knowledge. For instance, the malware should contain exploits that perform on the applications or Operating Systems (OSs) used in the plant. Malware might also need to communicate with the Programmable Logic Controllers (PLCs) and manipulate the values on the addresses linked to actuators to create a state that is not desired for day-to-day operations.

If the system knowledge needed to develop ICS tailored malware is kept secret, it will become harder for malware developers to attack a ICS.

## 1.3 Research question

To research the knowledge needed for a malware developer to create ICS malware, we first need to find the answer to the following question:

"What system knowledge is needed for a malware developer to create malware to infect and impact an Industrial Control System?"

The answer to this question will provide insight into what system knowledge is needed to create malware for ICS. ICS management staff can use this knowledge to define what information should be kept secret. A clear definition of what information should be kept secret can make it easier for ICS staff to prevent the information from leaking and will make it more difficult for malware developers to obtain the knowledge to create and test malware.

To answer the research question two sub-questions should be answered:

I "What system knowledge is required for malware to infect Industrial Control Systems?"

The way malware spreads can be seen as the repeated process of infection, privilege escalation and propagation by using exploits. An exploit will not work if an application, service or OS is not vulnerable to the exploit. Prior knowledge about the target systems, like which OS and programs are installed, is needed to write an exploit.

The answer to this research question will provide a list with the system knowledge information needed to infect an ICS.

II "What system knowledge is required for malware to impact the security of an Industrial Control System?"

The 'impact on security' can be categorized as Confidentiality, Integrity and Availability (CIA). Confidentiality is arguably not important to ICSs because usually no personal or secret data is handled. A data leak does not have direct consequences; it will not affect the productivity or safety of the system. The loss of Availability or Integrity can lead to direct consequences. Availability is often most important since a loss of availability means that it is no longer possible to control or supervise the system. Integrity is important because the loss of integrity could lead to a process that is controlled in an undesirable way and result in production losses, loss of equipment or endangerment of human safety. The 'impact' or 'impact on security' will refer to the loss of Availability or Integrity.

System knowledge is needed If malware manipulates the values on the addresses linked to actuators it might need to know what values to write, what addresses to write to, what protocol to use in communication and with what system it should talk.

Answering this research question will provide a list of system knowledge information needed to impact an ICS.

The 'knowledge needed' lists obtained from the two sub-questions will help us answer the research question.

## 1.4 Research method

To perform this research we set up a simulated environment that helped us determine the knowledge that was needed to develop malware.

The methodology to the research question, is:

1. Set up a virtual environment
   A virtual environment where an ICS is integrated with IT is set up to perform experiments. It includes a PLC, supervisory and control software with an Human Machine Interface (HMI) and a regular computer.

2. Create malware
   Create malware that can exploit a generic OS type vulnerability to spread towards an ICS and a HMI software vulnerability to infect the ICS. This step will provide lessons learned knowledge on the development of malware for ICSs.

3. Prepare a list of possible changes in the environment
   A list of possible changes in the environment will be made and used as test scenarios. This list will be reviewed and complemented by ICS and malware specialists with the use of a questionnaire. The questionnaire and the initial list of environmental changes is placed in Appendix A. The response to the questionnaire is in Appendix B. A code review is performed on the malware's code to find constant/static information, such as addresses, ports, implemented protocols and paths, and used to extract environmental changes from. The feedback obtained from the questionnaire increases the completeness and decreases bias of the list.

4. Change the environment or the malware according to a change mentioned on the list
   The environment will be changed to determine what changes affect the malware. If a change decreases the malware's capability to perform, it might indicate certain system knowledge is needed.

5. Determine the impact of the change
   For the first research question we check if the malware is able to infect, escalate and propagate. For the second question we check the malware's capability to impact availability and integrity.

The results of the first research question will create a 'System knowledge needed to infect ICSs with malware' checklist and it will be possible to extrapolate further requirements for general cases.

The impact of each change in the environment is a result and analysis will provide a list of the system knowledge needed to develop malware.

## 1.5 Contribution

The possibility to create ICS tailored malware with simulation software (MAlSim) has been researched [3, 21]. Several ICS attack models and scenarios have been defined and the attack scenarios were staged at a testbed [29]. Stealthy deception attacks were also investigated [2] where the attack was implemented at a physical canal. Several sources [2, 29] indicate that some system knowledge is needed to attack an ICS.

In the current state of the art the system knowledge needed to implement ICS tailored malware is not investigated.

To contribute to the state of the art, we define the system knowledge needed for ICS malware to:

i infect Industrial Control Systems.

ii impact availability or integrity of ICS components.

## 1.6 Thesis structure

Chapter 2 discusses the history of ICSs, typical architectures and differences between ICS and regular IT systems. Chapter 3 contains the details of the experimental environment, the used components to create the environment, and provides explanation about the physical environment. The chapter furthermore provides background information on malware, explains how the malware is written and the malware's exploits and payloads. Chapter 4 contains the results of the infection and impact phase of the malware. A summary is given of the results obtained from changing the environment. The complete results can be found in Appendix C. In Chapter 5 we analyze the results and give an answer to the research questions. In Chapter 6 we discuss related work. Chapter 7 concludes the whole thesis and mentions future work.

Chapter 2

# Industrial Control Systems

Industrial Control System are being used in many applications, for example in the production of food, drinks, electricity and oil. Other ICS control, elevators, traffic lights, rail traffic management and luggage conveyor belts at airports. They control physical processes with the use of hard-/software in an automated way. This chapter will explain the background of ICSs, how they work and provide insights into ICS security.

## 2.1 History

The first recorded application of a control method is believed to be Ktesibios's water clock in Alexandria (Egypt) dated ~250 BC. The principle behind a water clock is to let water drip at a constant rate from one reservoir to another reservoir. The second reservoir tells the time accordingly to its water level. This initial design is not entirely accurate since the water drip rate is higher when the first reservoir is fuller. This is where the control mechanism comes into the picture. A third reservoir and a 'float regulator' are introduced to fill the middle reservoir to an always full state. Figure 2.1 shows the mechanism. This control system ensured a constant pressure and a constant drip rate.

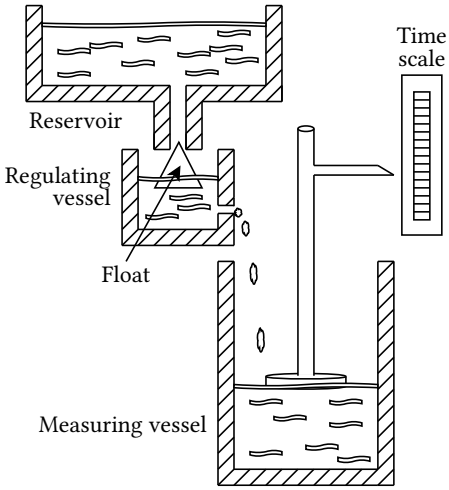



Figure 2.1: Ktesibios's water clock

Computer-controlled systems emerged during the late 1950's - begin 1960's [17]. In the 1990's ICSs looked different than the ICSs we know today; proprietary protocols, software and hardware were used and ICSs were isolated from the outside world. These properties made it impossible, or at least much harder, to perform computer-related attacks. Equipment safety, reliability and efficiency were important while cyber-security was not of importance [25]. Later, proprietary protocols, software and hardware were standardized to reduce costs. In the last decade businesses started integrating control systems with their business networks. The exchange of data between the control systems and business networks can enable cost efficient operations or increased production.

Supervisory Control and Data Acquisition (SCADA) systems are a type of ICSs and feature large scale processes that can include multiple sites and cover a large geographical area. Sometimes the term SCADA is used to refer to ICSs. This is because the term SCADA is generally better known by the layperson.

## 2.2 Control Theory

Control methods are used to control a variable, e.g., flow, amount, temperature, pressure, speed. Usually this variable is supposed to behave in a certain way over time. Disturbances can occur and they can change the output to an undesirable state. The output is used in the control loop as feedback, to return the output to the desired state. Figure 2.2 represents the control of a process.

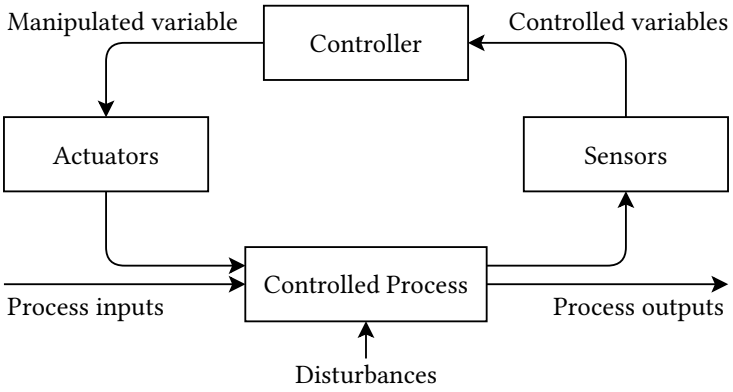



Figure 2.2: Process Control

With process control, disturbances are common. These can bring the current state in a state that is not equal to the desired state. The difference between those states is called the error. ICSs use sensors and actuators to control processes. One of the most widely used control feedback mechanisms is the Proportional-Integral-Derivative (PID) controller. With sensor measurements the present error P, the integral I (accumulations of past errors) and the derivative D (prediction of future errors) is calculated. To obtain the weighted sum, the P, I and D variables are multiplied by constants. The weighted sum is used to control the actuator (for example, motor, valve or robot-arm).

## 2.3 Architecture

The Purdue Model for Control Hierarchy [14] can be used as a model for typical control systems. An example of this model is shown in Figure 2.3. It segments devices into hierarchical functions. Every layer of the hierarchy is often referred to as level. There are six levels in total; level 0 to level 5 where levels 0 to 3 cover the manufacturing zone and levels 4 and 5 represent the enterprise zone. The hard- and software used for supervision and control can also be referred to as Operational Technology (OT) and can be found in the manufacturing zone. Similarly, the hard- and software used in the enterprise zone can be referred to as IT. The Purdue Model is covered bottom-up in the next subsections.

### 2.3.1 Level 0 - Processes

Level 0 is the lowest level where sensors, actuators and similar devices are involved in the basic control processes. Many kinds of sensors exist and can be used to measure flow, temperature, pressure, speed and other variables. These variables can be manipulated by actuators such as motors, valves, heating/cooling elements or fans. When sensors and actuators are combined they can be used to perform basic functions like moving objects to a certain place or position, maintaining pressure or maintaining a certain temperature. These devices sometimes operate in harsh environments and are not always easily accessible. To replace one or more devices would mean that a process can not be controlled for a (short) while and at least part of the manufacturing process should be stopped.

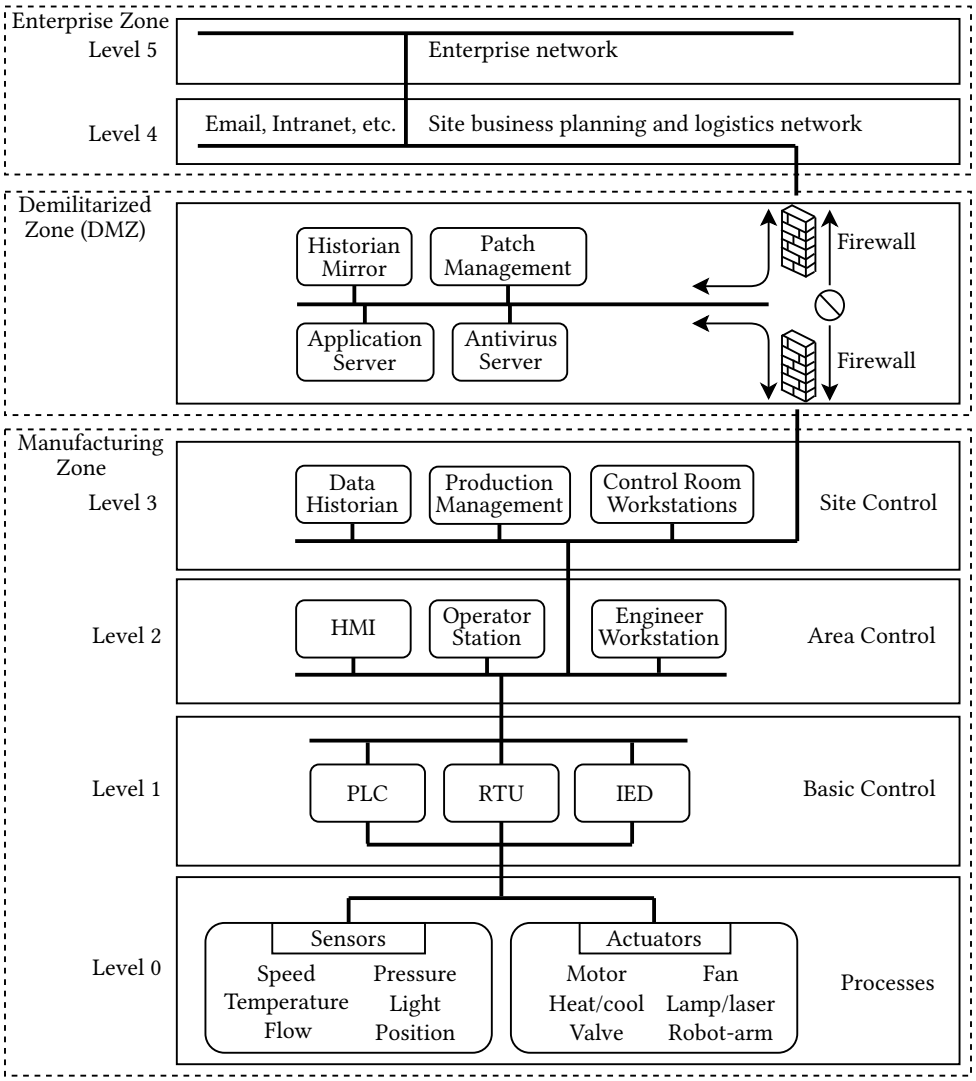



Figure 2.3: ICS architecture. Inspired by CISCO [6] and NIST [16].

### 2.3.2 Level 1 - Basic Control

This level consists of devices that control level 0 devices using discrete and analog signals. Three devices can be found in this layer; Programmable Logic Controllers (PLCs), Remote Terminal Units (RTUs) and Intelligent Electronic Devices (IEDs). They take the role of the controller shown in Figure 2.2. These devices can be programmed to control processes using multiple sensors and actuators. While their overall features are quite similar, they have differences.

The Programmable Logic Controller (PLC) is a microprocessor-based device often used in in SCADA and Distributed Control Systems (DCSs) [6]. It is usually highly configurable; they can be programmed to control a complex processes using logic, timing, counting and PID control. A PLC can used in a large range of applications as a generic solution and therefor the cost is relatively low, compared to a custom-designed component.

The Remote Terminal Unit (RTU), sometimes called a Remote Telemetry Unit, is often used for the purpose of remote control. It can come equipped with a wireless radio and has extensive communication options. This makes the RTU useful when a control system covers a large geographical area.

The Intelligent Electronic Device (IED) can perform control and monitor functions and communicate to SCADA systems with wired or wireless communication. It also includes protection functions, such as voltage regulators,

circuit breaker controllers, recloser controllers and capacitor bank switches. IEDs can include a HMI, such as a display and buttons for supervision and control.

### 2.3.3 Level 2 - Area Control

Level 2 represents an area within a site or a manufacturing process. The main elements are Distributed Control Servers (DCSs), Human Machine Interfaces (HMIs), operator stations, engineer workstations and switches.

A Distributed Control Server (DCS) can be used to control systems which are deployed at the same location. It supervises and controls level 1 controllers using communication protocols, such as Modbus or Fieldbus over Transmission Control Protocol (TCP). The DCS contains applications for product-, site asset- and performance-management.

The Human Machine Interface (HMI) is an interface for humans and machines to communicate. This is often a display or screen and buttons. The HMI is sometimes called a Man Machine Interface (MMI) or Graphical User Interface (GUI). It can be used to monitor or modify a state or a process. It can also be used to configure control algorithms and parameters in controllers. Typically it is used locally for one machine or piece of equipment. The HMI can also be used to display reports or historical data.

An operator station offers the same functionality as the HMI, but for multiple machines or pieces of equipment.

An engineer workstation is a regular personal computer where an engineer can supervise and control the area or manufacturing process. The engineering workstation can also be used to (re)program the logic of the PLC's. A PLC should either be connected to the engineering workstation with a programming interface to be programmed or have a SD-card or USB interface to load its new configuration from. A version of Windows is often used as OS for personal computers.

Switches are used to connect level 1 devices to level 2 and level 2 devices to level 3. The switch can contain a firewall, an Intrusion Detection System (IDS) or Port security. Port security prevents intruders to just physically plug a network cable into the network port and connect to the network. This is done by matching the Media Access Control (MAC) address of the connecting device with a known MAC address. A match will allow the device to connect. The communication link will be disabled if there is no match.

### 2.3.4 Level 3 - Site Control

This level supervises and controls the whole site. It contains a Data Historian, which is a centralized database for logging. A Data Historian logs all process information within an ICS so that the stored information can be accessed to support various analysis. A control room with engineer workstations, HMIs or other site level operation management is used by engineers to supervise and control the whole site. Production Management Software is used for scheduling and production reporting to manage the productivity of the site.

### 2.3.5 Level 4 and 5 - Enterprise zone

Levels 4 and 5 are not part of the control system zone. The enterprise zone contains regular IT equipment and is connected to the Internet. Level 4 is used for site business planning and logistics. Level 5 can contain all regular business activities. While important, these systems are not critical to the control system zone.

### 2.3.6 Demilitarized zone (DMZ)

The Demilitarized zone (DMZ) acts as a divisor between enterprise environment and the manufacturing environment so that data can be shared. It can segment control of the enterprise environment and manufacturing environment. No traffic should pass the DMZ directly, but should always travel through it. Firewalls should be configured to enforce this. The DMZ adds another layer of security. For example, the historian mirror in the DMZ synchronizes at an interval with the historian in level 3. If a level 4 or 5 application needs information from

the historian it can now connect to the historian mirror in the DMZ. There is no longer a need to connect to the Manufacturing zone. This reduces the number of connections from the Enterprise zone to the Manufacturing zone.

## 2.4 Comparison of ICS and IT

As stated in Section 2.1 'History', ICS are resembling IT systems more than before. They have their similarities but also their differences. One of the biggest differences is that ICSs directly affect the physical world with actuators to, for instance, produce a product or provide a service. If the control system is not able to produce a product or provide a service, it will likely result in immediate financial loss or other damages. Availability is therefor often seen as the most important aspect. Another important aspect for ICSs is safety in the sense of physical safety and equipment safety. The loss of integrity could mean that a controller would control a process in an undesirable way and lead to production losses, loss of equipment, endangerment of human safety or environmental damages.

In Table 2.1 several key-differences between IT systems and ICS are pointed out. An important item to note is change management. The availability and integrity requirements make ICSs hard to patch. Many ICSs are not often patched or not patched at all [18]. Since components can have a lifetime up to 15-20 years, it is possible to find old unpatched systems. This creates a big contrast to IT systems where new equipment is bought every few years and (security) patches are often automatically installed. Old and unpatched systems may contain known vulnerabilities that can be exploited by malware.

| Category | Information Technology System | Industrial Control System |
|---|---|---|
| Risk Management Requirements | Data confidentiality and integrity is paramount<br>Fault tolerance is less important - momentary downtime is not a major risk<br>Major risk impact is delay of business operations | Human safety is paramount, followed by protection of the process<br>Fault tolerance is essential, even momentary downtime may not be acceptable<br>Major risk impacts are regulatory non-compliance, environmental impacts, loss of life, equipment, or production |
| Architecture Security Focus | Primary focus is protecting the IT assets, and the information stored on or transmitted among these assets. Central server may require more protection | Primary goal is to protect edge clients (e.g., field devices such as process controllers) Protection of central server is also important |
| Unintended Consequences | Security solutions are designed around typical IT systems | Security tools must be tested (e.g., off-line on a comparable ICS) to ensure that they do not compromise normal ICS operation |
| Time-critical Interaction | Less critical emergency interaction<br>Tightly restricted access control can be implemented to the degree necessary for security | Response to human and other emergency interaction is critical<br>Access to ICS should be strictly controlled, but should not hamper or interfere with human-machine interaction |
| System Operation | Systems are designed for use with typical operating systems<br>Upgrades are straightforward with the availability of automated deployment tools | Differing and possibly proprietary operating systems, often without security capabilities built in<br>Software changes must be carefully made, usually by software vendors, because of the specialized control algorithms and perhaps modified hardware and software involved |
| Communications | Standard communications protocols<br>Primarily wired networks with some localized wireless capabilities<br>Typical IT networking practices | Many proprietary and standard communication protocols<br>Several types of communications media used including dedicated wire and wireless (radio and satellite)<br>Networks are complex and sometimes require the expertise of control engineers |
| Resource Constraints | Systems are specified with enough resources to support the addition of third-party applications such as security solutions | Systems are designed to support the intended industrial process and may not have enough memory and computing resources to support the addition of security capabilities |
| Change Management | Software changes are applied in a timely fashion in the presence of good security policy and procedures. The procedures are often automated | Software changes must be thoroughly tested and deployed incrementally throughout a system to ensure that the integrity of the control system is maintained. ICS outages often must be planned and scheduled days/weeks in advance. ICS may use OSs that are no longer supported |
| Component Lifetime | Lifetime in the order of 3-5 years | Lifetime in the order of 15-20 years |
| Access to Components | Components are usually local and easy to access | Components can be isolated, remote, and require extensive physical effort to gain access to them |

Table 2.1: Differences between IT and ICS. Obtained from NIST Special Publication [16].

# Chapter 3 Experimental environment

This chapter describes the environment and malware used in the tests. A simulated environment was used because the developed malware will try to impact availability and integrity, the properties that are most important to ICSs. It is not feasible to test malware on a real working ICS. The scenario, set-up and the components of the environment will be discussed here.

A literature study on ICS simulation testbeds was performed. Several testbeds were found, such as the one described by Béla Genge et al. [13] based on Emulab to create cyber components and Simulink to create physical processes. The testbed is used to measure the impact of attacks against the cyber and physical parts of the system. Another testbed described by Fovino et al. [10] uses an experimental test-bed, deployed in Italy, to simulate attack scenarios against SCADA systems. The experimental testbed contains a physical power plant emulator with real PLCs and sensors. Reaves et al. propose a virtual testbed for industrial security research [23] build using a network simulator called 'ns-2'.

The purpose of this research is to research the knowledge needed to develop malware to infect and impact an Industrial Control System. For that we need to measure the effect of a change in the environment on the malware and be able to determine if the malware can adapt to different environments. We need an experimental environment that can be changed and adapted according to the changes we would like to make in the environment. We also need real ICS software for the malware to exploit.

The testbed described by Béla Genge et al. [13] is not very suitable because Emulab has disabled several Windows facilities/services (e.g., firewall) that we plan to use as one of our change in the environment. While all Windows 7 changes are documented, this is not the case for the Windows XP changes [1]. The testbed described by Fovino et al. [10] can not be used because setting up a real physical testlab with PLCs, sensors and actuators is outside of the scope. The testbed proposed by Reaves et al. [23] is not suitable because it does not provide full support for the TCP protocol (e.g. no RESET segments) [2]. We can obtain unreliable results if any of the unimplemented TCP functionality is used by the HMI or malware to communicate with the PLC during our tests.

We looked at existing ICS testbeds but we can also consider other ICS software and simulators that are available. We chose this approach because it provides a fast way to start the project and is available for free. The HMI software provided a graphical demo project that we use to get started. We extend the setup according to our needs and described it in the following sections.

## 3.1 Scenario

The experimental environment will represent a part of the chemical plant which mixes and stores chemicals in large tanks. The tanks are connected by pipes and pumps are used to pump chemicals from one tank to another. The first tank has an entry point with a valve to add chemicals while another tank has an exit point with a valve for chemicals to flow to the next process. This is shown in picture 3.1. The chemicals should be in exactly the right ratio before they leave the final tank. An ICS is used to control the processes using sensors and actuators. Sensors read the chemical levels in the tanks while the actuators control the state of the valves and pumps. A PLC is able to communicate with the sensors and actuators as well as with other systems. The PLC contains programmable logic to perform basic control of the mixing process.

Engineers can supervise and control these processes using a HMI which queries the PLC for sensor values and graphically displays it a screen. The software also enables engineers to control the actuators through the PLC. It

[1] https://wiki.emulab.net/wiki/Emulab/wiki/Windows
[2] http://www.isi.edu/nsnam/ns/ns-limitations.html

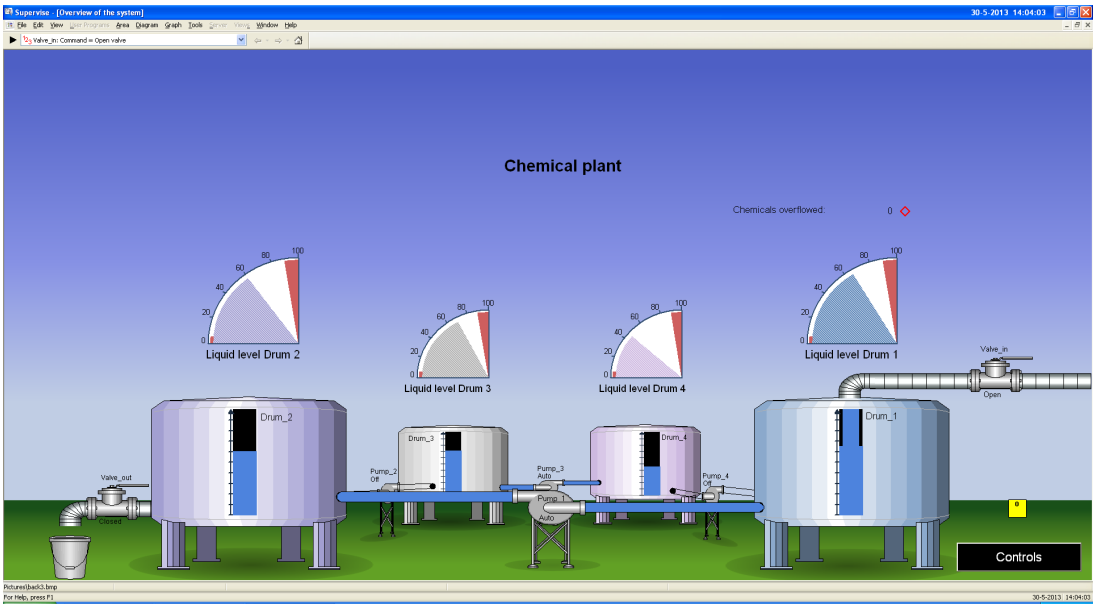
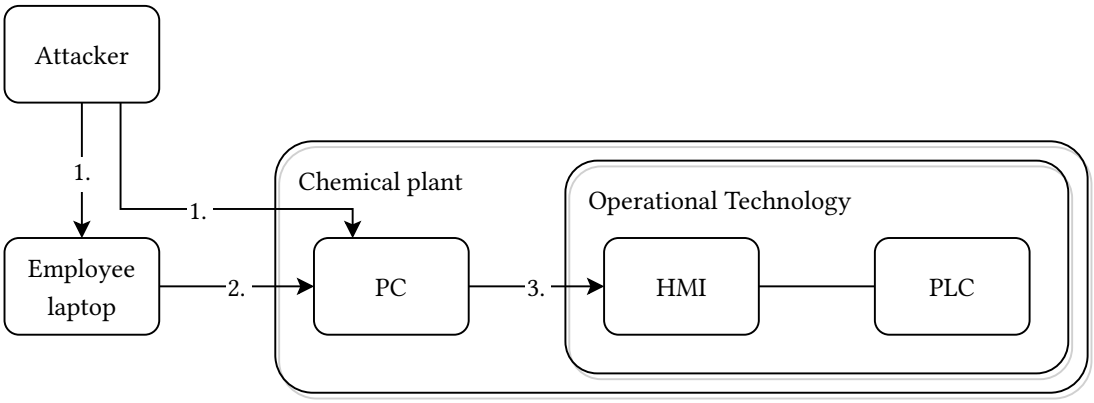



Figure 3.1: Overview chemical plant HMI

is important to note that software influences the physical world using actuators. Supervision and management is done at the plant in a control room.

One day an employee wants to supervise the plant from home using his work-laptop. At the plant he sets up a remote desktop service and configures his laptop to connect to the service. Consider the case where the employee goes home and his laptop gets infected with malware. When the employee uses his credentials to log-in to the remote desktop service the malware records his credentials. With these credentials malware can also log in and spread to the plant.

Malware can also spread to the plant in other ways. USB sticks or other USB devices are being used to transport configuration files of the logic of the PLC. Another way to spread would be to infect a USB stick with these files on it. The malware will spread once the USB stick is inserted in a machine in the plant. These infection paths are shown in Figure 3.2.




1. Infected USB device
2. Infected USB device, network share or physical movement of laptop
3. Infected USB device or HMI software exploit

Figure 3.2: Infection path

Inside the plant malware spreads to a machine where it can communicate with the PLC and intercept traffic

between the HMI and the PLC. It can tell the PLC to manipulate valves and pumps to mix the chemicals in an undesirable way or overflow a tank with chemicals while telling the control software that everything is normal. This way malware can influence the physical world and create production losses, environmental damages or even potentially affect human safety.

## 3.2 Components

The mentioned scenario is recreated in a virtual environment where every component is running in its own Virtual Machine (VM). The VMs are run with VirtualBox [3]. A virtual network is made between the machines according to Figure 3.2. The 'Employee laptop' and 'PC' described in Figure 3.2 will run Windows 7 SP0. An older version of Windows is used for the OT part of the plant because software updates are not often updated or patched or not patched at all [18]. Windows XP SP3 is used for the VMs in the OT part of the plant. The components described in this section are used to represent the described scenario.

### 3.2.1 HMI software

Interactive Graphical SCADA System (IGSS) version 9 is used [4] to supervise and control the environment. IGSS is developed by 7-Technologies (7T). 7-Technologies has been acquired by Schneider Electric in August 2011. A newer version of IGSS, version 10 is released in September 2012. This version is most likely not adopted everyone because of the slow rate of which updates are planned for ICSs as mentioned in Section 2.4. IGSS provides: a tool to design HMIs which in turn can be used to supervise and control a SCADA system; support for many communication protocols; and functionality to set preventive maintenance jobs. Warnings and alarms are defined in the HMI designer. A graphical effect or other actions are configured on a warning or alarm. It is also possible to generate rapports and monitor communications.

*Configuration*

The HMI software that represents the scenario is shown in Figure 3.1. The case where all drums are overflowing is shown in Figure 3.3. In the figure we can see that the level indicators on each drum indicate a full drum. On the right we see the number of chemical units that have overflowed. Also note the orange square in the right lower corner. This object is linked to the alarm module of the HMI. The alarm module shows all errors and warnings. The square's color shows the type of error or warning and the number indicates the amount.

As visible in figures 3.1 and 3.3, each drum has two indicators which indicate the chemical level: A bar placed on the drum and an additional pie indicator. It is possible to configure warnings and alarms for each drum. Every drum is set to give a warning when it is almost empty or almost full. The almost empty warnings are configured when the drum reaches a chemical level of 5 units or below and the almost warning is configured when the drum can only store less than 10 units more. The pie indicator marks those areas with red. In Figure 3.3 all drums are full, i.e., the chemical level reaches 100 units. A full drum is not desirable because any addition of chemicals will result in an overflow. An alarm is configured to fire during this condition. It is also possible to set an 'animated symbol' during warnings or alarms. This way it is possible to show, hide, blink or change the color of an existing element. An animated symbol is used to create an 'overflow animation' which is linked to the 'drum is full' alarm. We've used this to create the overflow animation.

*Vulnerabilities*

ICS environments are often equipped with old and unpatched systems and software as revealed in survey [18] and discussed in Section 2.4. This opens the possibility for attackers to use existing exploits to target ICSs.

[3] http://www.virtualbox.org/
[4] http://igss.schneider-electric.com/products/igss/

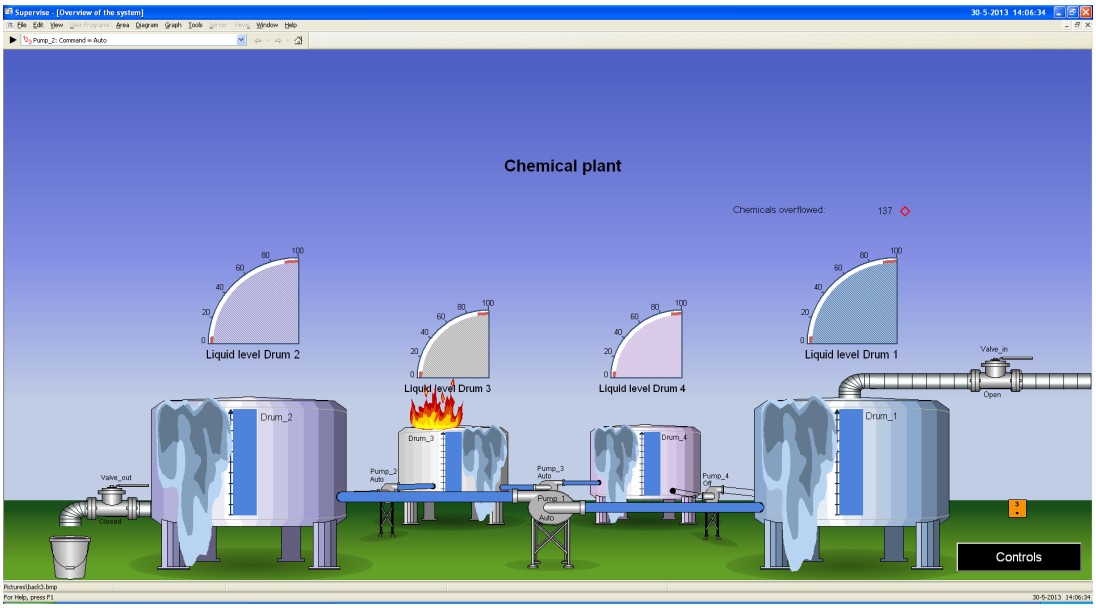



Figure 3.3: HMI representation of the chemical plant with drums overflowing.

IGSS version 9 has several remote vulnerabilities. It is possible to infect the HMI by exploiting two of these remote vulnerabilities.

The first service IGSSdataServer runs on port 12401 and is responsible for the synchronization of data, including the HMI description, between the IGSS server and the operator stations. This functionality is also used when a setup is created with a primary and a secondary hot-backup server. The graphical HMI description is stored in several files. The service synchronizes the HMI description files with remote read, create, write, rename and delete commands. With a directory traversal attack it is possible to perform those actions for any file on the system [5]. The protocol used by the service is TCP/Internet Protocol (IP) with a proprietary application level protocol.

Another vulnerability is the directory traversal vulnerability in the Data Collector dc.exe which can be accessed from port 12397. The service collects and calculates the data which is shown on the HMI. It also contains an EXE opcode to run an executable with a CreateProcessA() function as a new thread. Arguments can be given but are optional. This functionality is not protected against directory traversal attacks. By exploiting this vulnerability any file can be executed on the remote host [6]. TCP/IP with a proprietary application level protocol is used.

All versions $\leq$ 9.00 of IGSSdataServer and versions $\leq$ 9.00 of dc.exe are vulnerable. The exploit details state that they are tested with IGSS versions 9 and lower that run on Windows XP, Windows 7 and Windows Server 2003 / R2. Service packs do not have any influence on the exploit.

### 3.2.2 Programmable Logic Controller (PLC)

The PLC is simulated using a Modbus PLC simulator program called Mod_RSsim [7]. A running instance is shown in Figure 3.4. The simulator shows coil outputs, digital and analogue inputs and registers that can be written to or read from. These are shown in a table with 10 addresses per row and the values can be shown in decimal, hexadecimal, as a float or character string. The PLC simulator supports multiple connections and is able to communicate using Modbus/RS-232, Modbus/TCP and DF1 (configured as an Allen Bradley Slave or a simple master that simulates a JOY SCC386). A communication monitor can show all send and received messages.

The values of the coils and registers can be manipulated by the PLC simulator to simulate a physical environment.

[5] CVE-2011-1565
[6] CVE-2011-1566
[7] `http://www.plcsimulator.org/`

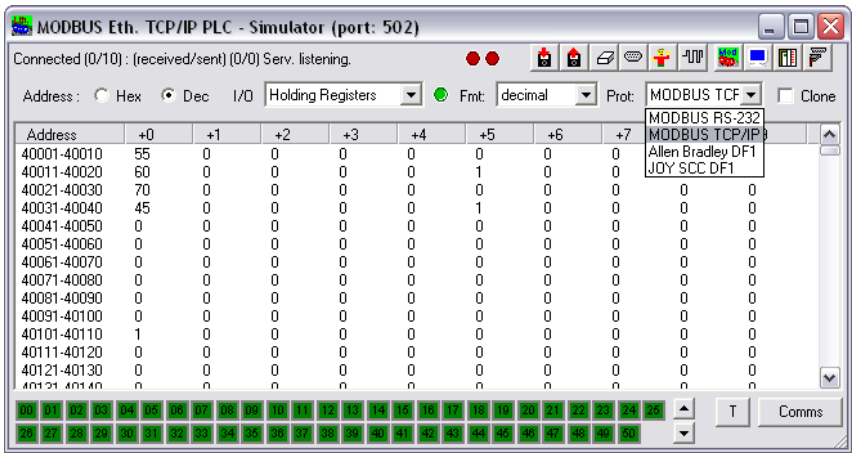



Figure 3.4: Mod_RSsim - PLC simulator

The PLC simulator can be automated with a Visual Basic script that is executed every 500ms. Some functions have been provided for the script to interact with the simulator. For instance the functions GetRegisterValue and SetRegisterValue can be used by the VBscript to read register states from the PLC and write register values back. We use this functionality to simulate the physical processes of the chemical plant. The physical processes described in Section 3.1 are simulated with a Visual Basic script and described in the next subsection.

## 3.3 Physical processes

This subsection will explain the physical processes of the chemical plant example that is used throughout our experiments. The in Section 3.1 described part of the chemical plant contains four drums and four pumps to pump chemicals from one drum to another in a clockwise fashion. A valve regulates the addition of chemicals to drum 1 and another valve regulates the amount of chemicals that leave the process through drum 2. This process is shown in Figure 3.5.

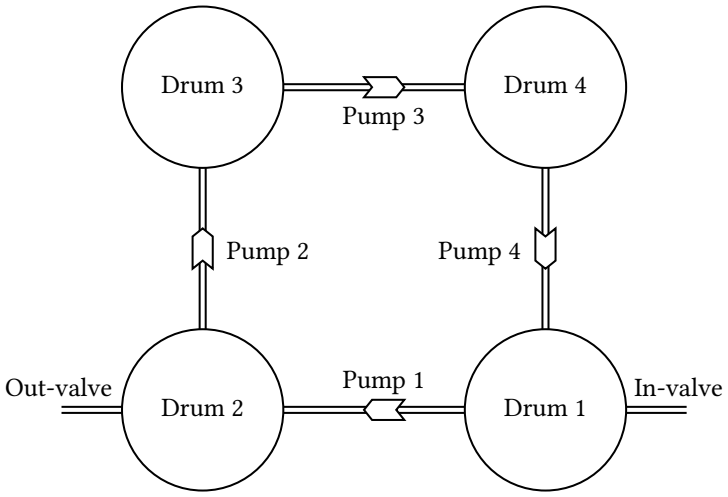



Figure 3.5: Overview chemical plant

The amount of chemicals a drums is able to contain ranges from 0 to 100 units where 0 is completely empty and 100 means full. If chemicals flow to a full drum, it will result in an overflow. An undesired state or 'critical state' is reached when the drums are completely empty or overflowing. The amount of spilled chemicals is recorded

and shown on the HMI. The actuators are simulated once every time the script is executed. Pumps have three settings: 'On', 'Off' and 'Automatic'. The setting 'On' will pump chemicals with a speed of 2 units per script execution to the next drum. When the pump is set to 'Off' it will not do anything. The setting 'Automatic' only pumps chemicals to the next drum when the chemical level of the current drum is higher than 80 and the next drum has a lower chemical level than the current drum. Valves have two settings: 'open' and 'close'. When a valve is open it will enable a chemical flow with a speed of 1 unit per script execution. These physical processes will be simulated by the Visual Basic script and can be found in Appendix D.

The default physical environment state is the following; both in- and out-valves are closed, drum 1 has a chemical level of 55, drum 2 has a chemical level of 60, drum 3 has a chemical level of 70 and drum 4 has a chemical level of 45. Pumps 1 and 3 are set to 'auto' and pumps 2 and 4 are off. This state has been chosen because it is a stable state; nothing changes. This way, it is possible to get repeatable results since the time between resetting the default physical environment state and the execution of the malware is not important.

The described environment will be referenced from now as the 'default environment' or 'clean environment'. We write malware that is able to infect and impact this setup according to the specifications described in the next section.

## 3.4 Malware

This section covers different existing IT and ICS malware and describes the intentions and structure of the malware that is developed for this environment.

### 3.4.1 ICS targeted malware

Stuxnet was different from other malware as it directly targeted Operational Technology. Stuxnet is a computer worm type of malware that targeted Iran's uranium enrichment facilities to sabotage the ICSs [9]. It was discovered in June 2010 and believed to be created by the U.S. and Israel [24] to stop or at least delay Iran's nuclear program. Stuxnet includes a few known exploits, four zero-day exploits, the first ever PLC rootkit, a Windows rootkit, peer-to-peer updates, a command and control center and other functionalities [9]. The uranium enrichment facility's centrifuges were targeted by reprogramming the PLCs to repeatedly adjust the spinning speed to very slow and faster than normal [9]. This process will make it look like the centrifuge broken down because the centrifuge was of low quality. Stuxnet destroyed more than a thousand centrifuges.

### 3.4.2 IT targeted malware

The following malware have not directly targeted the OT part of the ICS but mainly targeted the IT part or were used for espionage.

*Duqu*'s main purpose is to gather information and assets from entities such as industrial infrastructure and system manufacturers. This is to obtain information such as design documents to more easily perform future attacks. Duqu is found in September 2011. It is a Remote Access Trojan and it does not contain code to replicate itself or any code related to ICSs. Duqu was written by the same authors as Stuxnet (or those with access to the code) [28].

*Flame*, also known as Flamer and sKyWIper, is a modular malware used for targeted cyber espionage in Middle Eastern countries. It can record audio, screenshots, keystrokes, network traffic and Skype conversations [8]. Flame uses five encryption methods and SQLite to store structured information [7]. To reduce the probability of detection it also determines what antivirus software is installed to customise its own behavior. The in May 2012 discovered Flame is very large for a malicious program: it is 20 megabytes large. It allows other attack modules to be loaded after infection. The source code of at least one module of Flame was used in Stuxnet. This proves that the Stuxnet and Flame developers have cooperated at least once during early development [15].

*Night Dragon* is a coordinated covert and targeted cyberattacks against global oil, energy and petrochemical companies that has been conducted since November 2009 [26]. Night Dragon is a Trojan backdoor with the

goal to obtain sensitive competitive proprietary operations and financial information. These attacks primarily originated from China and involved social engineering, exploitation of Windows OS vulnerabilities, Microsoft Active Directory Compromises and the use of remote administration tools [26].

*Shamoon* (Disttrack) is a modular computer virus that attacks Windows OSs and is used for cyber-sabotage in the energy sector. Shamoon is build up out of three components: a Dropper, which is the main component and source of the infection; a Reporter, which reports infection information back to the attacker; and a Wiper that corrupts files on an infected machine and overwrites the Master Boot Record. The wiper component is responsible for the destructive functionality; it replaces an existing driver with another digitally signed driver that enables user-mode applications to read and write to disk [1].

### 3.4.3 Custom ICS tailored malware

With all those different malware out there, we can ask ourselves: 'Why not reuse Stuxnet to answer the research question?'. It is possible to download the reverse engineered Stuxnet source code. At first it seems like a good idea, because it saves time. On the other hand, we need to familiarize ourselves with the reverse engineered Stuxnet code. The code is not well documented and while most functions have understandable names, most variable names are not representative; they are a combination of a letter and a number. We need to know the code well enough it we want to modify it. This takes considerable time. Another counterargument is that Stuxnet was programmed to target one type of plant and will not activate unless it detects that it has reached its target. To recreate that plant might also take a long time. In the end it will likely take less time to develop malware ourselves. This way we can implement the functionality we need and we have more freedom to choose and modify our environment.

We will develop malware and test if it still spread and affect the integrity and availability of the ICS when the experimental environment changes. The malware's goal is to spread to the chemical plant and create a security impact at the plant. When the malware infects a regular IT system it will not try to impact the system, since it is not the goal of the malware. This action would do more harm than good as it increase the chance of getting detected. Initially the malware only scans for other systems and spread. Only once the malware detects that it reached its target, it will try to create an impact on the security of the plant. This strategy is programmed into our malware. The malware's architecture is shown in Figure 3.6.

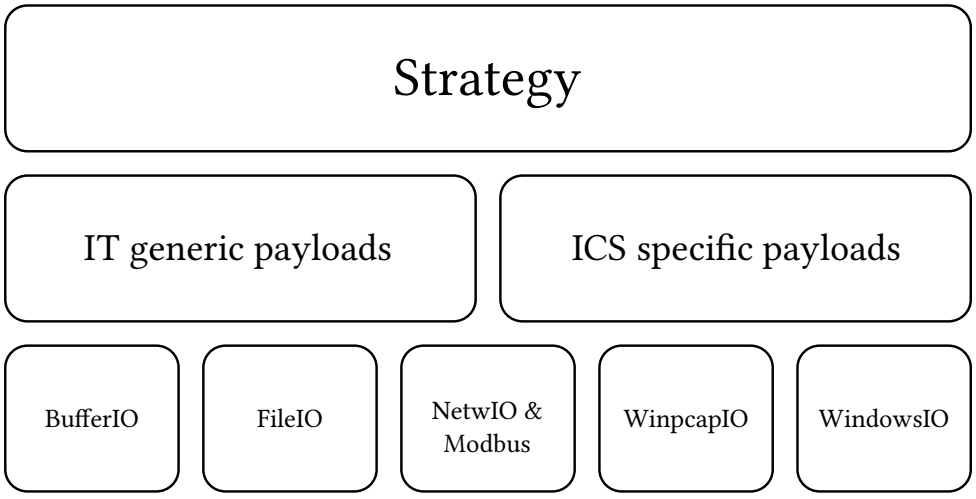



Figure 3.6: Malware architecture

The malware works according to the scenario described in Section 3.1 and infection path shown in Figure 3.2. An attack graph is given in Figure 3.7 and described here. The initial infection can be caused by a targeted attack. The attacker will either infect a USB device or an attachment and target a plant employee or someone related. When an infected USB device is used as attack vector, it can be left in front of the home or car of a plant employee or be given to a family member. When the USB stick is plugged in a computer it will automatically

launch the malware and infect it (steps 1b and 2b). An infected attachment can be emailed to the employee. When the infected attachment is opened, an exploit will launch the malware (steps 1a and 2a). Now the malware has

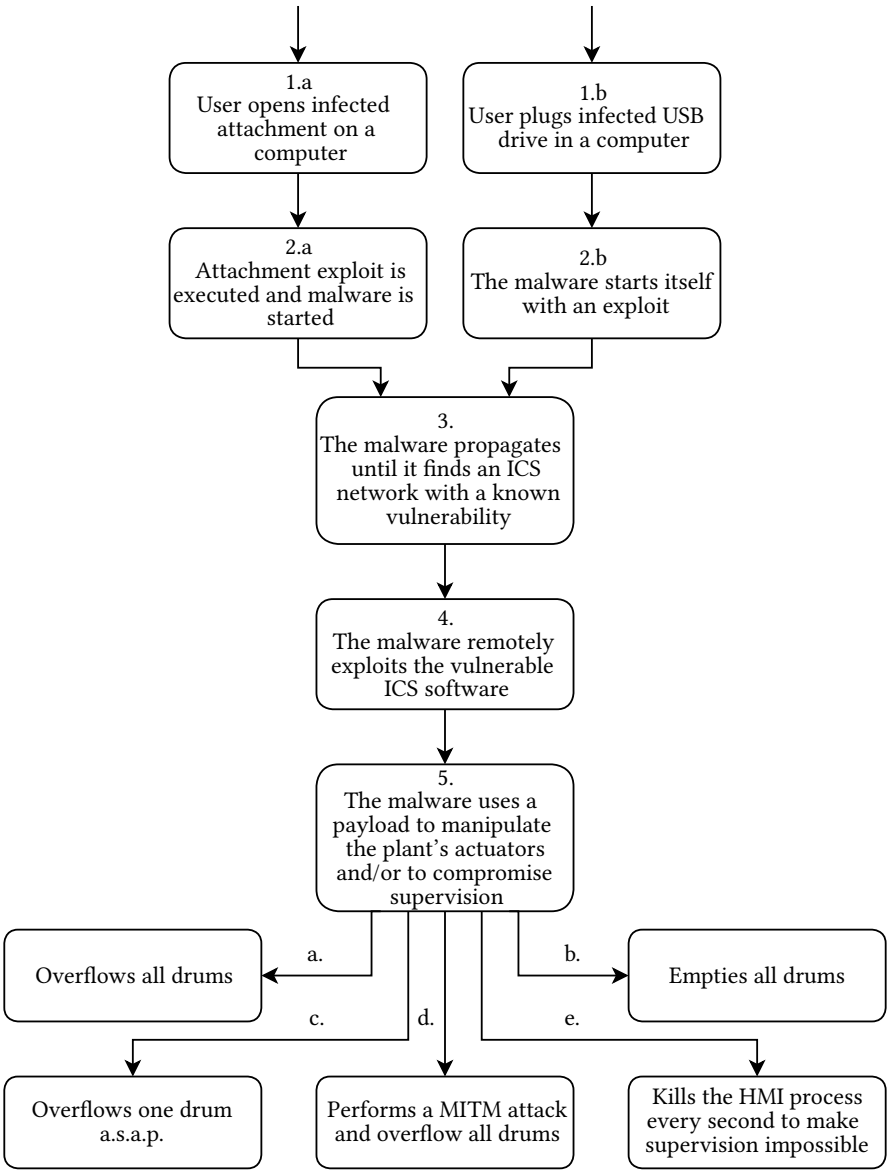



Figure 3.7: Malware timeline

infected a computer, it will try to spread to the plant. It will infect USB drives and network shares until it infects the plant-employee's laptop or a machine connected to the plant (step 3). If the malware detects a connection to the ICS with software it will perform a remote exploit to infect the machine running the vulnerable ICS software (step 4). The malware will finally execute one of the payloads (step 5) to manipulate the actuators in the plant (i.e., the valves and pumps) and compromise the supervision capabilities of the HMI. These payloads impact the availability and/or integrity of the plant. The payloads can stop the HMI from displaying the correct information, stop the HMI from working, cause production losses, chemical spillage and possibly environmental damages. A more detailed explanation of the created payloads is given later in this chapter.

The malware is programmed in C with Microsoft Visual C++ Express Edition. The Microsoft Visual Studio solution consists of two project; the first for the malware and the other project builds the Dynamically Linked Library (DLL) used a the payload to infect USB drives and network shares.

The project settings of both projects should be set to use the Windows XP platform toolkit so our malware is

compatible with Windows XP. The runtime library setting should be set from 'Multi-threaded DLL' to 'Multi-threaded' so that most libraries (DLLs) will be included in the executable. This will make our malware less dependent on the other system's libraries. During the development phase we have tested basic changes on the 'Employee laptop' and the 'PC' as defined in Section 3.1. These changes include; removing admin privileges, changing OS version, changing network interface configurations and changing IP addresses. This was mainly done to iron out bugs but also to asses the malware's stability.

*Design decisions*

Arguably the biggest design choice had to be made during the implementation of the Man-in-the-Middle (MITM) attack. During our attack we would like to let the HMI believe that all systems work as normal. This is done by performing a MITM attack; we send traffic to the HMI and make it look like it came from the PLC and vice versa. When performing a MITM attack, we need to forge (spoof) packets. This makes the target believe a packet is originated from a specific machine. Since we want to spoof Modbus/TCP traffic, we need to spoof TCP and IP fields. These spoofed packets should contain the source IP address of another machine and we also need to set other TCP and IP fields. This information is usually filled in automatically when a network socket is created and it can not be changed. To set these fields we need a raw socket. If we have a raw socket we can manipulate all fields of the packet.

C does not support raw sockets for TCP connections [8]. This means that there is no easy way of spoofing TCP traffic. It is possible to implement custom network sockets (in Assembly or if possible in C) but this would take quite some time and research. Another option was to use the tool 'winpcap' [9]. Winpcap is a link-layer network access for Windows environments, it allows applications to capture and transmit network packets bypassing the protocol stack. It is used as capture and filtering engine of many known network tools. Winpcap can be used to change any packet field necessary. We decided to use the winpcap functionality for spoofing packets because a determined attacker will likely be able to spoof packets. A determined attacker can create the functionality to send raw packets over the network, embed the winpcap installer into the malware or create the same functionality as winpcap provides.

Another design decision was whether to use a scanner or not. Currently our malware uses one remote exploit. It is possible to run the exploit on every IP address in the subnet's range. This would require quite some time since the timeout of every connection is set to 30 seconds. A scanner would enable solution that is faster and more scalable with the number of exploits. The method we're looking for should be fairly accurate, preferably fast and simple. We will therefor not scan every port of every machine but just one port per machine. A simple ping sweep would do the job here as it meets all requirements. We have used a simple ping scan to detect all machines in the network.

The malware is supposed to execute its final payload only when it has reached the plant. The malware determines if it reached the plant by checking if the infected machine had HMI software (IGSS) installed. This means that the malware could also have triggered on nontargets that used the same HMI software.

*Payloads*

Eight payloads were developed, two to infect the plant and six more to create an impact on the security of the plant.

PAY-1. The first payload is used to infect USB devices and shared folders. It uses the same vulnerability [10] in the Windows' explorer program as Stuxnet did. When explorer shows a folder to the user, it will load the files and, if visible, their icons. A shortcut file (.lnk) is a Windows shortcut to another file. It will usually display the icon of the file it links to, but their icon can be changed more dynamically than any other file type. When a specially crafted icon of a shortcut is shown, an arbitrary library (DLL) can be specified to be loaded and executed.

[8] `http://msdn.microsoft.com/en-us/library/windows/desktop/ms740548(v=vs.85).aspx`
[9] `http://www.winpcap.org/`
[10] The 'ms10_046 shortcut icon DLL loader' vulnerability

A library was developed to find and execute our malware. This library will be seen as part of the malware. When the malware infects a shared medium, it will copy itself to the shared medium and create an infected shortcut. The shortcut's icon references to the malware (library part). In short; when a folder with the infected shortcut is shown, part of our malware will be loaded and executed. The malware will run as part of the exploited explorer process with the same rights. Our malware infects USB devices and network shares with shortcut files and it also copies itself there. This payload is used to infect USB devices and network shares in step 1b, 2b and 3 of Figure 3.7. This exploit will later be referenced as the 'shortcut exploit' or 'shortcut vulnerability'.

PAY-2. This payload is used to infect the HMI in the environment. It exploits the remote vulnerabilities in the IGSS services mentioned in Section 3.2.1. The IGSS data server service contains a directory traversal vulnerability which can be used to perform file operations. The data collector service has a directory traversal vulnerability which can be exploited to run any executable present on the remote machine. These vulnerabilities are remotely exploited by the malware to copy and execute itself on the target machine. This payload is used in step 4 of Figure 3.7.

PAY-3. The payload's goal is to put the system in a critical state; it will overflow all drums. It impacts the integrity of the plant by manipulating actuators. The payload sends Modbus/TCP commands to the PLC to open the 'in-valve', close the 'out-valve' and set the pumps to automatic. The drums will gradually fill and eventually overflow. No MITM attack is performed so the HMI will show the real state of the control system during the attack. The harmful state is written to the PLC periodically (e.g., every second) or every time an engineer manipulates the state of the pumps or valves.

PAY-4. The previous payload can be modified to empty all drums. This payload is a copied version of payload PAY-3 with the exception that it empties all drums.

PAY-5. To create impact as quickly as possible, a payload can be used that only overflows the fullest drum. This payload checks the chemical levels in each drum and overflows only the fullest drum by manipulating the pumps and valves.

PAY-6. While the previous payloads do not hide their actions, it is also possible to be more stealthy. A modified MITM attack can used to intercept traffic between the PLC and HMI. It is not possible to execute a standard MITM attack because only two devices are involved in the attack. The first machine is running the HMI and the malware. The second machine simulates the PLC. The attack is shown in Figure 3.8. First a

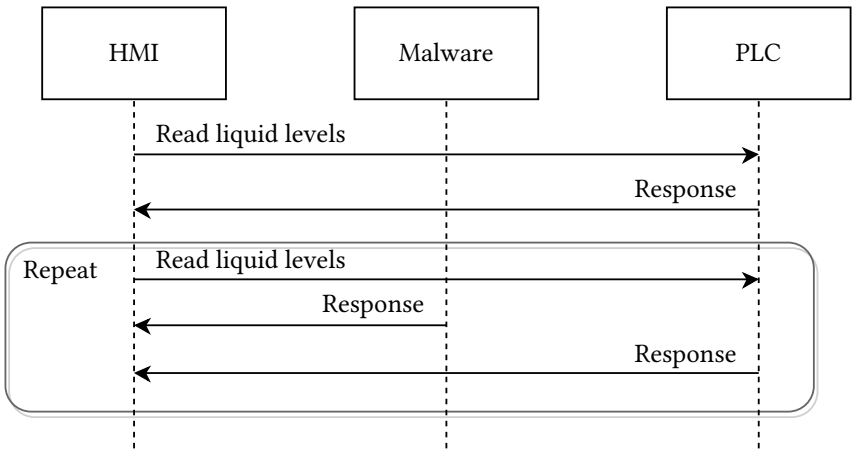



Figure 3.8: MITM type 1

response packet is captured. Then this packet is modified and replayed every time a new request is made. This attack relies on the assumption that the malware's packet arrives at the HMI before the PLC's packet do. This payload can be combined with PAY-3 or PAY-4 to also influence the physical processes during the MITM attack.

PAY-7. This is another variant of a MITM attack. It is based on the lack of authentication in the Address Resolution Protocol (ARP). This protocol is used to convert network layer addresses to link layer addresses, like IP addresses to MAC addresses. All conversions are (temporarily) stored in a table-like cache. We can use this protocol to trick machines to send network traffic to the wrong destination. This can be done by spoofing an ARP packet. If the malware had administrator privileges, a permanent ARP entry can be added directly. This payload first captures a response packet, so it can be resend later on. A packet (FIN) is send to the PLC to close the connection. The PLC acknowledges (ACK) the end of the connection. But since the HMI had no intention of closing the connection, it will keep it open. Finally an ARP packet is spoofed to set the HMIs MAC address as the PLCs. All traffic destined for the PLC will be send to the HMI. The malware will respond to every request. This attack is shown in Figure 3.9. The default ARP

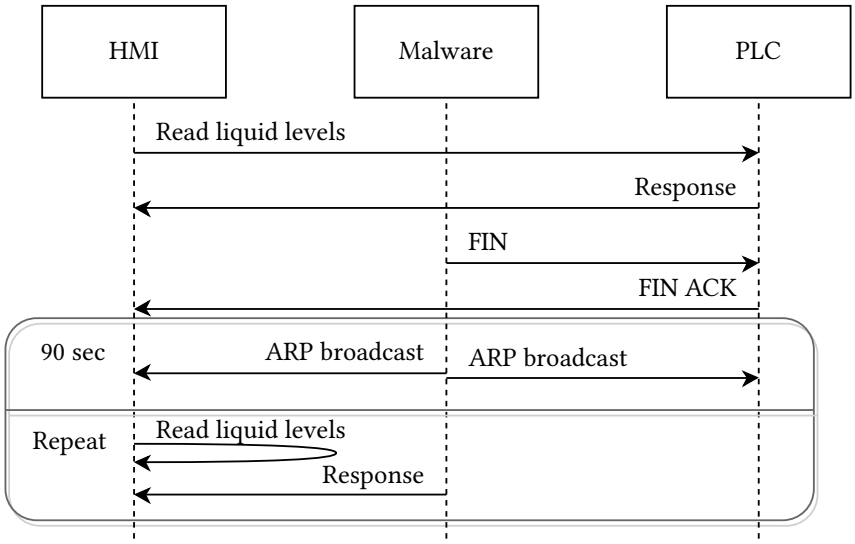



Figure 3.9: MITM using FIN and ARP

cache expiration time for unused IP addresses is 2 minutes. The malware therefor spoofs ARP packets at least once every 2 minutes to ensure the connection to the PLC will not be reestablished. This payload can be combined with PAY-3 or PAY-4 to influence the physical processes during the MITM attack.

PAY-8. IGSS provides a utility that is able to terminate all IGSS processes, including the HMI. It is useful if a process hangs and therefore prevents other functionality from opening or functioning. The program is named killigss.exe and can be found in the IGSS program directory. Killigss can also be used to perform a Denial of Service (DoS) attack on the HMI. The HMI will take several seconds to start. If killigss is executed every second, it would be very hard for the HMI to start up. The engineers will not be able to use the HMI to supervise the plant anymore.

*Other functionality*

The bottom layer in the architecture in Figure 3.6 contains modules that provide functionality for exploits and payloads to;

- Create, fill and modify buffers and create buffer patterns.
  The exploits used in metasploit that are written in Ruby have easy ways to create buffers and an easy way to create a pattern which can be used to find and calculate return addresses more easily. These functionalities has been developed in C and used to our exploits.

- Create, read and write files to/from disk.
  A simple library is created to perform regular disk IO.

- Connect and communicate with a Modbus/TCP device.
  Libmodbus [11] is integrated into the malware to provide this functionality.
- We can make regular network connections
  Regular network connections are used to scan the local network for machines and to determine if specific ports, used by vulnerable services, are open.
- Sniff and spoof traffic.
  The winpcap library is used to sniff traffic and send raw TCP packets. A layer on top of that provides the functionality to interpret the raw traffic, modify and spoof it.
- Obtain information about the current machine.
  We can obtain information such as the name and IP address of the system, current logged in user, if the current user has admin rights and we can read from and write to registers.

In the next chapter we obtain results according to the methodology described there.

[11] `http://libmodbus.org/`

# Chapter 4 Results

Now that the environment is setup and the malware is implemented, it is possible to understand what knowledge is needed to develop malware for the environment. In this chapter two types of results are described: the first result describes how the malware infects and impacts the security of the environment. This environment is called the 'default environment' or the 'clean environment'. Secondly we will change the environment and determine if the malware is still able to infect and impact the security of the environment. The process of changing the environment will be called an 'environmental change'.

If an environmental change reduces or diminishes the effect of the malware, it does not immediately imply that the developer lacked system knowledge. The developer might have made a design decision that another developer would have made differently. Some design decisions do not support certain changes in the environment. Those design decisions will be revised to mature the malware without introducing new system knowledge. For instance, when the malware was first developed, it scanned the network for vulnerable machines with a ping scan. The ping scan was blocked when firewalls were enabled. The design decision was revised by implementing an ARP scanner. Sometimes the malware can not adapt to the changing environment, not because it was due to a design decision, but because the developer lacked system knowledge. For example, the malware developer needs to know what communication protocol is used between the HMI and PLCs. If the malware does not support the protocol that is used, it might not be able to impact the integrity of the processes. This means that sometimes multiple attempts are required to determine if an environmental change can reduce or diminish the effect of the malware. Those 'attempts' are called 'tests'. The first test will determine the effect of the environmental change on the malware. If the malware fails to successfully execute a payload we either modify the malware and perform another test or we determine that system knowledge is needed. If a test succeeds, we can test other variations or decide to stop.

The rest of the chapter is structured as follows: First, the methodology for obtaining the results is defined. Then a description on how the malware infects and impacts the security of the default environment is given. Lastly we discuss if the malware was still able to infect and impact the security of the environment after an environmental change. These results will help answer the research questions in the next chapter 'Analysis'.

## 4.1 Methodology

To get reliable and repeatable results, we need a methodology to perform environmental changes. The methodology includes: 1) allowed user-interaction; 2) what is monitored and 3) which payloads are used. The methodology will be linked to the previously defined scenarios in sections 3.1 and 3.4 to show these steps are conform.

The method to test the impact of the payloads on the clean and changed environment is explained here.

1. For each test we start with a clean environment because a well defined starting point will enable us to have repeatable results. We also do not want malware from the previous test to interfere with the new test.
2. We implement an environmental change before our first test. This step will not be performed when we test the impact of the malware on the clean environment. For every next test we either change the environment or change the malware without introducing new system-knowledge in the malware.
3. When we initiate a test we perform limited user-interaction; we only run the malware. The tests are according to the infection path defined in Section 3.1, Figure 3.2 and the malware timeline mentioned in Section 3.4.3, Figure 3.7. The step where the malware is manually executed simulates the opening of the infected attachment in scenario 1a and 2a in Figure 3.7. The step where the USB drive is used represents scenario 1b and 2b in Figure 3.7. To start the malware we:

a. execute the malware once from command line or by double click or
b. insert a USB drive. If autoplay is not configured, navigate with explorer to the root of the drive.

No further user-interaction with the system is performed after one of the scenarios is initiated.

4. The systems is monitored according to how the malware is supposed to behave.
   (a) If the USB drive is used as an infection medium we check if the malware launches using the shortcut exploit. If the malware launches, payload PAY-1 is successful.
   (b) We check if the malware executes the remote exploit (PAY-2) to propagate to the HMI.
   (c) We check if the malware infects the HMI. If the malware starts, payload PAY-2 has executed successfully.
   (d) The malware is supposed to choose and execute one of its payloads.
   - If the malware terminated the HMI process, payload PAY-8 has executed successfully.
   - The malware should able to modify the PLC address(es) used by the actuator(s). Payloads PAY-3, PAY-4 and PAY-5 are successful if the state of at least one actuator is changed.
   - Payloads PAY-6 and PAY-7 execute successfully if the HMI displays the PLC state that spoofed by the malware (and the response from the PLC is discarded or no response from the PLC is received).

The results of these tests can be found in Section 4.2.

When we determine if an environmental changes reduced or diminishes the impact of the malware we perform the steps mentioned above. We execute step 2 to change the environment before we test the effect of the malware. For some environmental changes we felt the need to perform multiple tests. For some environmental changes we needed to perform up to four tests to learn the effect of the environmental change on the malware. For every additional test we either modified the environment or malware and the changes were documented. If the malware could not (completely) adapt it often meant new system knowledge was needed. We noted if the malware did not behave exactly as in the clean environment.

Several payloads are considered successful under the same conditions. Therefor we will not test all payloads when we test the effect of the environmental changes. For every environmental change we check if the malware;

1. can spread using both infection methods; with the HMI exploit (PAY-2) and the USB exploit (PAY-1);
2. can modify the state of the actuators (PAY-3);
3. is able to perform a MITM attack (PAY-7) and
4. can terminate the process of the HMI repeatedly (PAY-8).

For some environmental changes we felt the need to test more or all payloads. We have noted which additional payloads are tested. This way we can reduce a big amount of tests in a way that will not influence our results.

Now that we have defined our tests, we can start performing them.

## 4.2 Default environment

Eight payloads were developed to infect and impact the security of the system. The effect of the execution of each payloads will be discussed here.

RESULT-1. Payload PAY-2 scans for USB drives and network shares and infects them with crafted shortcuts. The exploit is triggered if a user views the contents of the infected folder and the malware is executed. When an infected folder is accessed through command line it will not trigger the exploit since no graphical information is processed.

RESULT-2. Payload PAY-2 exploits the vulnerable services of the HMI by coping and executing itself to another machine. The exploited services can not be disabled in the HMI interface. Several services can be enabled and disabled in the project properties. The exploited services are listed there, but can not be disabled since they are checked and their checkbox is disabled. If the services are manually terminated, they are restarted automatically by the HMI master program. During our tests, there was not a single instance when the HMI service exploits failed. These properties combined make the infection phase of this exploit very reliable.

RESULT-3. Payload PAY-3, fills and overflows all drums. The payload keeps writing a harmful state to the PLC. It will open the 'in-valve', close the 'out-valve' and manipulate the pumps. This state is graphically represented in Figure 4.1. This state will fill every drum simultaneously to the rim and eventually overflow.

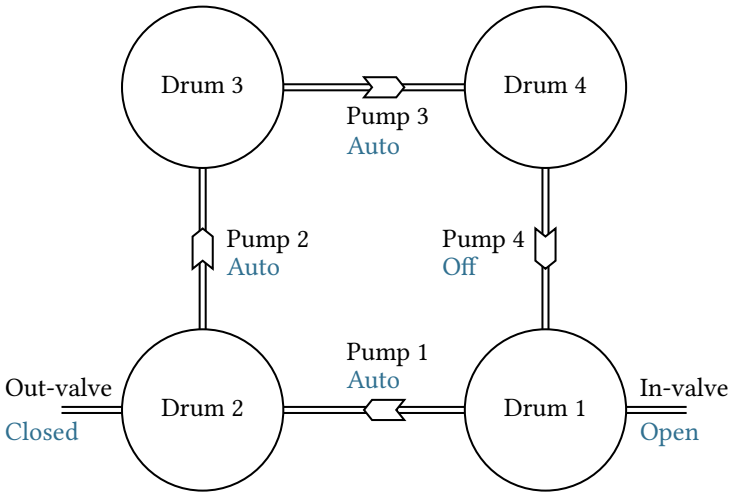



Figure 4.1: A state where all drums are filled until they overflow.

RESULT-4. Payload PAY-4, has a similar logic but will try to empty all drums. The state it will try to maintain can be found in Figure 4.2.

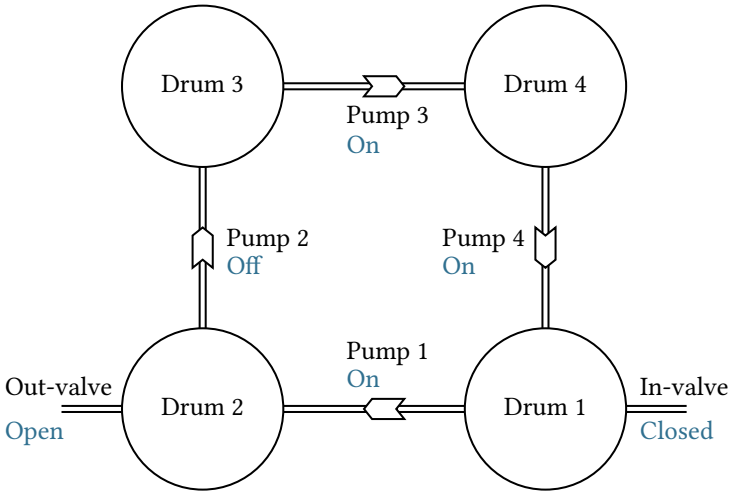



Figure 4.2: A state where all drums are emptied.

RESULT-5. Payload PAY-5 overflows the fullest drum as fast as possible. It will read the chemical level in all drums, choose the fullest one, open the in-valve and close the out-valve. For example, if we would

want to overflow drum 3, we would turn pumps 1, 2 and 4 on and pump 3 off. This example is shown in Figure 4.3. It will also rewrite the harmful state to the PLC if it detects user-intervention in the

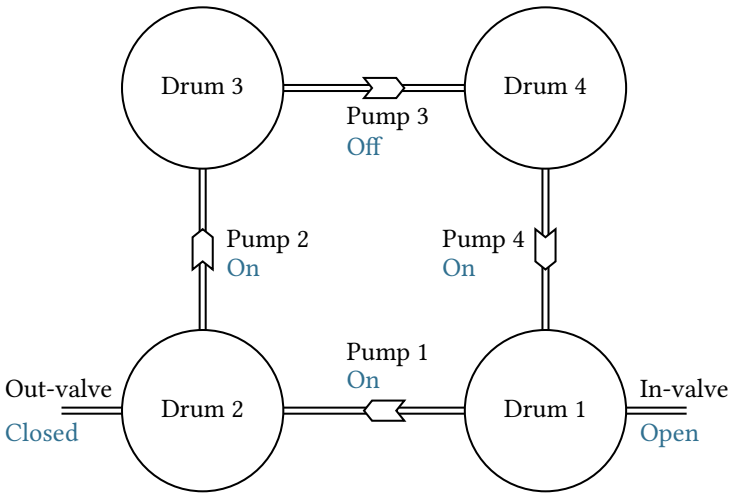



Figure 4.3: The state where drum 3 is filled as quickly as possible.

same manner as the previous payloads.

RESULT-6. Payload PAY-6 contains a MITM attack and is generally used in combination with PAY-3 or PAY-4. It also responds to the HMI's request. If the malware responds faster than the PLC, then the PLC's response is treated as a replay and therefore discarded. The simulator used for the PLC has an option to specify the response latency. If the response latency is set to 0ms it will respond faster than the malware. If the response latency is increased to 10 - 16ms then the malware will gain a better chance to successfully spoof responses. A response latency greater than 35ms is enough to let the malware spoof responses faster than the PLC for over an hour. It should be noted that the simulated environment has a low network latency (of less than 1ms).

RESULT-7. Payload PAY-7 contains a MITM attack based on ARP-spoofing and can be used in combination with the overflow of all drums or empty all drums payloads, PAY-3 or PAY-4 respectively. Since this attack poisons the ARP cache, it effectively leaves the HMI machine unable to communicate with the real PLC until the spoofed ARP cache entry times-out. New entries to the ARP cache are timestamped and will timeout if not reused for 2 minutes. If an entry is used, it receives two more minutes of lifetime. When this payload is used alone, it is an availability type of attack because even if the malware does not respond or exits then the HMI will have to wait until its ARP cache entry times-out. During that time it is not possible to supervise or control the system.

RESULT-8. Payload PAY-8 executes the killigss.exe program every second. This is an availability attack (DoS) on the HMI software. The HMI software process is terminated before it is able to load the HMI. When this payload runs it is not possible to supervise or control the plant. A restart of the infected machine will make the HMI usable again. But the malware is able to create an entry in one of the startup registries to start every time the computer is turned on.

In the previous tests, we have used quite some system knowledge during the implementation. This means we will not be able to reliably determine which knowledge is needed. We change the environment to determine if the malware can adapt to those changes. System knowledge is needed if the malware can not adapt to the environmental change because the developer lacked system knowledge rather than due to a design decision. An overview of the results is given in the next section.

## 4.3 Environmental change results

The environmental changes are split into four categories: HMI changes, network changes, PLC configuration changes and physical process changes. These categories are graphically represented in Figure 4.4. Note that all

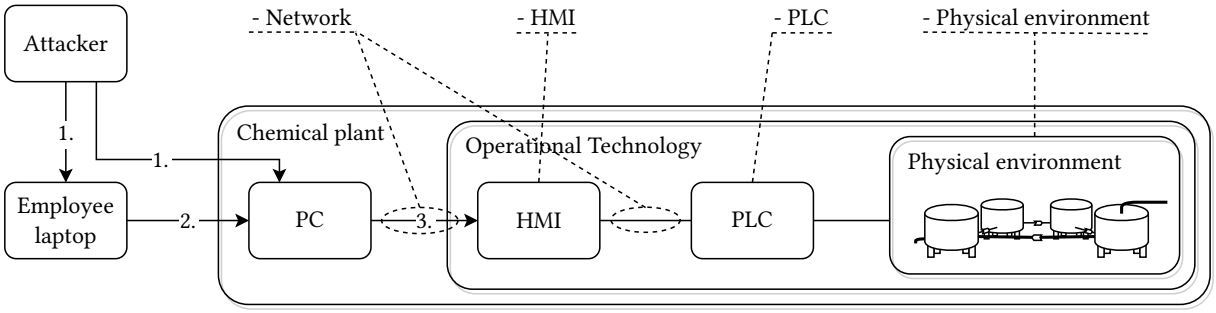



Figure 4.4: Environmental change categories for infection and impact

parts of the setup have an environmental change category except for the PC and Employee laptop. Regular IT tests, covered in the HMI changes, have been performed on the PC or Employee laptop during the development of the malware. This was to get the malware working reliably and the results were covered in Section 4.2. The PC and Employee laptop will not be tested again.

Section 4.3.1 contains an example of a complete environmental change. All other environmental changes are summarized per category. The complete environmental change results can be found in Appendix C.

### 4.3.1 Example of an environmental change

This section covers an example of an environmental change. This environmental change tests if knowledge is needed about the addresses used by the PLC and its sensors/actuators. The example is quoted from Appendix C and can be found by searching for environmental change PLC-1. During this environmental change we first defined what changes were made, why this test is relevant to determine the knowledge needed to develop malware and a hypothesis. The environment is changed to determine if the malware can still infect and impact the security of the plant. In the first test the malware was able to perform an availability attack but it could not attack the integrity of the processes anymore.

> Change the register addresses used by the PLC
> *Reason for testing*
> The register addresses of the sensors and actuators are hardcoded in the malware. The malware knows which addresses are associated with the actuators. If we change the addresses used by the sensors and actuators, then the malware will still use the old addresses. If the malware writes to the wrong addresses (i.e., the addresses not used by actuators) then the physical process(es) will not be affected. This would, for example, mean that the malware would be unable to overflow or empty the drums.
>
> *Hypothesis*
> The malware will be able to spread to the HMI machine but the impact will be different if all register addresses are changed, then the malware will manipulate the wrong register addresses. This implies that the malware is unable to manipulate actuators and thus target the integrity of the processes. It will still be possible to attack the availability (payload PAY-8 will still succeed).
>
> *Test 1 - process to change the environment*
>
> - Change the PLC simulation script to use the same addresses, but with an offset of 1000.

- Change the HMI project definition to use the new addresses.

*Test 1 - results*

- This change did not influence the infection.
- The malware was able to compromise the supervision capabilities of the HMI using the MITM payloads but was unable to manipulate the actuators. The malware could still target availability by terminating the HMI process repeatedly with payload PAY-8.

*Test 2 - process to change the malware*
We will test if the malware can adapt to register changes by maturing it and without introducing new knowledge. We developed functionality that analyzes the environment for a specified time. With this information we can create an impact on the integrity of the environment. The malware will analyze the environment until it has analyzed a predefined number of packets.

- When the HMI is infected, the first step would be to listen to every connected PLC. The malware allocates and initializes the variables that are used to analyze the traffic.
- The malware analyzes the traffic. It looks at the memory registers that have been written and the minimum and maximum values that are read. Finally, it records a response packet which will be used later to spoof responses.
- The malware writes to every writeable memory register either its minimum or maximum recorded value.

*Test 2 - results*

- This change did not influence the infection.
- The malware is able to write either the highest or lowest recorded value to each writeable PLC register. It is not possible to define the impact on the ICS in advance, because the state created by the malware is random.

*Lessons learned/System knowledge needed*
If we do not know what registers are used to read from and write to, we need to analyse the traffic or write to all addresses. During the tests, we analyzed the traffic between the PLC and HMI and determined which values were needed to be spoofed and which addresses were connected to actuators.

The impact phase relied on the assumption that the smallest or largest values are potentially the most dangerous values for actuators. With the knowledge gained from analyzing the packets during runtime we were able to create a random potentially dangerous state. Another similar possible implementation could be to write random, zeros or maximum values to the PLC. An increased probability to create impact can be achieved if we know that larger values are more/less likely to create impact than small values. More consistent results are created when the registers are set to the highest recorded value.

The malware analyzed the traffic when the PLC communicated to the HMI through Modbus/TCP. The malware might not be able to analyze the traffic, if the traffic is encrypted or when another protocol or interface is used. We can not determine the possibilities for the other cases yet. We determine if it is also possible to analyse serial connection traffic when we connect the PLC to the HMI with a serial connection in environmental change PLC-5. ”

For the second test we looked into the possibility to adapt the malware to the environment by making another design decision. We modified the malware so it is able to perform integrity attacks again in the second test. The notation of the environmental tests ends with a few comments on the learned lessons and an indication if knowledge about the system was needed to pass these tests. The process is repeated for all environmental changes. Appendix C contains the details of all environmental changes, including this one. The analysis functionality created during this test will later be used in other environmental tests, mainly to handle changing physical environments.

### 4.3.2 Summary

The results of all environmental changes are summarized and formatted into tables per category. The example used in Section 4.3.1 is formatted into a table entry. This is shown in Table 4.1. The summary describes the environmental change, states if system knowledge is needed and clarifies possible notes in the table. The malware was able to infect the HMI in both tests. This is noted with a '✓' mark. In the first test it was not possible to perform integrity type of attacks but the availability attack succeeded. This is shown with a '✗' and a note that can be read in the text describing the table. In the second test we noted that we used an analyzer to perform an integrity attack.

> The environmental change, outlined in Table 4.1, is described as follows:
>
> The malware needed a way to determine which addresses were used when we changed memory addresses for the sensors and actuators (PLC-1). [a] Otherwise it would write to the wrong memory addresses. [b] The malware analyzed network traffic to determine the addresses that were used by the actuators and which values were usable.

| | Test 1 | | | Test 2 | | |
|---|---|---|---|---|---|---|
| | Infect | Impact availability | Impact integrity | Infect | Impact availability | Impact integrity |
| PLC-1 | ✓ | ✓ | ✗[a] | ✓ | ✓ | ✓[b] |

> If the malware was able to infect a host a '✓' character is noted, otherwise a '✗'. The '✓' character was used in the impact phase if the malware was able to impact the availability or if it could manipulate at least one actuator, otherwise a '✗' is noted. Notes are explained in the text.

Table 4.1: Example result of a change in the PLC.

The tables indicate the results of the performed tests of all environmental changes. For the infection and impact phases we noted if the change had an effect on the malware. Machines can be infected by either an infected USB drive with a shortcut exploit or by a vulnerability in the HMI with the HMI exploit. For some environmental changes we wanted to perform multiple tests. If a cell is left blank, no test was performed. A character in superscript refers to an explanation that is given in the text.

*Human Machine Interface changes*

We tested the effects of several changes in the OS as well as changes in the HMI settings and software changes. The result of the environmental changes are shown in Table 4.2. Some environmental changes did not reduce the effect of the malware, such as changing the HMI's IP address (HMI-3); project paths (HMI-7); update intervals (HMI-8) or running the HMI as unprivileged user (HMI-9). Enabling or disabling Data Execution Prevention (HMI-4) or Address Space Layout Randomization (HMI-5) on the HMI also did not affect the malware.

A few of these environmental changes initially uncovered design decisions that could be improved. [f] When the HMI's IP address (HMI-3) was changed, the malware could initially not determine if the HMI was installed. It therefore did not realize it found the target. The malware checked for an registry entry that was only present on the account which had installed the software. This was solved by choosing another registry entry. [c] The scanner initially scanned for the first 20 addresses in the Local Area Network (LAN)'s IP range. The malware now scans the whole range.

[d] Not all tests could be performed for environmental change (HMI-5) because the HMI software could not be forced to use Address Space Layout Randomization (ASLR). No tests were possible for environmental change (HMI-13) because the HMI does not support a backup channel for a PLC.

Other environmental changes had limited effect on the malware. When autoplay was configured, no user interaction would be required after the insertion of the USB device to infect the machine. [e] If autoplay was not configured, the user had to navigate to the infected folder (HMI-6). When we added a second HMI to overview the plant, it did not affect the infection phase or the attacks on availability or integrity (HMI-10, HMI-11). Multiple HMIs that supervise the same PLCs might reduce or diminish the effect of the MITM payloads. If the malware can not communicate or infect the HMIs that supervise the same PLCs, then the HMIs will show different states. This can be a trigger for employees to examine the plant which can expose the malware. [g] An analyzer is used to determine which PLCs are supervised by the infected HMI when every HMI only supervises its own part of the plant (HMI-12). Having winpcap installed on the HMI will create an easy way for the malware to spoof packets and perform MITM attacks. If winpcap is not installed and the malware wants to perform a MITM attack, it would have to install it or find another workaround (HMI-14). [h] The malware initially relied on the MITM attack but it was not possible to spoof packets and the MITM payloads stopped working. [i] The malware can determine if it can perform a MITM attack and still impact the integrity and availability of the ICS.

The environmental changes that had effect on the malware were the following: [a] A firewall can be used to block several scan types and MITM attacks [1], such as payload PAY-7. This was seen in environmental change HMI-1. Another scanner was developed which was not blocked by the firewall. When we changed the version of the OS to a newer one, it affected the shortcut exploit because the vulnerability was patched (HMI-2). [b] Windows 7 SP1 patched the shortcut vulnerability. When the control system network is not connected to the outside world (air-gapped), it is harder to infect the plant (HMI-15). [j] The malware was not able to remotely exploit the plant; it has to travel on a physical medium (e.g., USB device, CD/DVD, laptop) through the air-gap.

| | Infect | Impact availability | Impact integrity | Infect | Impact availability | Impact integrity | HMI exploit |
|---|---|---|---|---|---|---|---|
| | **Test 1** | | | **Test 2** | | | **Test 3** |
| HMI-1 | ✗[a] | ✓ | ✓ | ✓ | ✓ | ✓ | |
| HMI-2 | ✓ | ✓ | ✓ | ✗[b] | ✓ | ✓ | |
| HMI-3 | ✓ | ✓ | ✓ | ✗[c] | ✓ | ✓ | ✓ |
| HMI-4 | ✓ | ✓ | ✓ | | | | |
| HMI-5 | ✓ | ✓ | ✓ | [d] | [d] | [d] | |
| HMI-6 | ✓[e] | ✓ | ✓ | | | | |
| HMI-7 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| HMI-8 | ✓ | ✓ | ✓ | | | | |
| HMI-9 | ✓ | ✗[f] | ✗[f] | ✓ | ✓ | ✓ | |
| HMI-10 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| HMI-11 | ✓ | ✓ | ✓ | | | | |
| HMI-12 | ✓ | ✓ | ✓[g] | | | | |
| HMI-13 | | | | | | | |
| HMI-14 | ✓ | ✓ | ✗[h] | ✓ | ✓ | ✓[i] | |
| HMI-15 | ✗[j] | ✓ | ✓ | | | | |

If the malware was able to infect a host a '✓' character is noted, otherwise a '✗'. The '✓' character was used in the impact phase if the malware was able to impact the availability or if it could manipulate at least one actuator, otherwise a '✗' is noted. Notes are explained in the text.

Table 4.2: Result from changes on the HMI machine.

[1] Gratuitous ARP packets can be blocked by the firewall

*Network environmental changes*

For these environmental changes, we changed the network properties and send noise and other traffic over the network to determine if it affected the malware Table 4.3 is described here. We added simulated traffic to the network (NET-1), simulated an instable network connection (NET-2), introduced traffic from a legitimate source (NET-3) and replayed previously captured traffic (NET-4). These environmental changes did not affect the malware; it kept working as usual. One network change affected the malware; [a] when we put the control system in another virtual LAN, it behaved in exactly the same way as when the plant was air-gapped (NET-5).

| | Test 1 | | |
|---|---|---|---|
| | Infect | Impact availability | Impact integrity |
| NET-1 | ✓ | ✓ | ✓ |
| NET-2 | ✓ | ✓ | ✓ |
| NET-3 | ✓ | ✓ | ✓ |
| NET-4 | ✓ | ✓ | ✓ |
| NET-5 | ✗[a] | ✓ | ✓ |

If the malware was able to infect a host a '✓' character is noted, otherwise a '✗'. The '✓' character was used in the impact phase if the malware was able to impact the availability or if it could manipulate at least one actuator, otherwise a '✗' is noted. Notes are explained in the text.

Table 4.3: Result from changes in the network.

*PLC configuration environmental changes*

The environmental changes are based on adjusting settings of the PLC. Table 4.4 describes the outcomes of the tests. Changing the IP address or the port of the PLC did not reduce the effects of the malware (PLC-2, PLC-3). [c] The malware initially only searched for PLCs listening to the default Modbus port. The malware's filters were changed to support non-default ports. Some changes influenced the workings of the malware, albeit limited. A low response latency caused one of the MITM payloads (PAY-6) to not function (PLC-4). The malware needed a way to determine which addresses were used when we changed memory addresses for the sensors and actuators (PLC-1). [a] Otherwise it would write to the wrong memory addresses. [b] The malware analyzed network traffic to determine the addresses that were used by the actuators and which values were usable. The malware was able to communicate to multiple PLCs after developing a way to detect connected PLCs and by using the earlier developed analyzing functionality (PLC-7). Environmental change PLC-6 was not tested because it was similar to Modbus over serial port.

[d] When we changed the environment to use Modbus over serial port instead of Modbus/TCP, the malware could not find the PLC anymore (PLC-5). The functionality to communicate with devices over serial connections was not yet build in. We added support for serial port connections to enable the malware to communicate to the PLC to perform integrity attacks. [e] Only one application can connect to the serial port at a time, so the malware had to terminate the HMI driver to allow itself to connect to the PLC. [f] Since only one application can connect to a serial port at a time, it was not possible to perform MITM attacks or listen to regular traffic between the HMI and PLC. Therefor it was not possible to analyze the traffic. This required the malware to guess the serial port settings. This was feasible since the number of combinations are limited. The malware needed prior knowledge to perform integrity attacks. Since only one application could connect to a serial port at a time, it was possible to perform another DoS attack; one that disconnected the HMI from the PLC and occupied the connection to the serial port. This way the HMI could not connect to the PLC anymore.

| | Test 1 | | | Test 2 | | | Test 3 |
|---|---|---|---|---|---|---|---|
| | Infect | Impact availability | Impact integrity | Infect | Impact availability | Impact integrity | Impact integrity |
| PLC-1 | ✓ | ✓ | ✗[a] | ✓ | ✓ | ✓[b] | |
| PLC-2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| PLC-3 | ✓ | ✓ | ✗[c] | ✓ | ✓ | ✓ | |
| PLC-4 | ✓ | ✓ | ✓ | | | | |
| PLC-5 | ✓ | ✓ | ✗[d] | ✓ | ✓ | ✗[e] | ✓[f] |
| PLC-6 | | | | | | | |
| PLC-7 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

If the malware was able to infect a host a '✓' character is noted, otherwise a '✗'. The '✓' character was used in the impact phase if the malware was able to impact the availability or if it could manipulate at least one actuator, otherwise a '✗' is noted. Notes are explained in the text.

Table 4.4: Results from changes on the PLC.

*Physical process environmental changes*

When we tested how changes in the physical environment affected the malware, we mostly changed the workings or the quantity of the sensors and actuators in the environment. The result of the environmental changes are shown in Table 4.5. The amount of pumps or drums (PHY-1, PHY-2) did not have an effect on the malware. The amount of liquid that the drums can hold or the speed at which the liquids flow is also irrelevant to impact availability or integrity (PHY-4, PHY-6).

When the amount of valves was changed, the malware behaved the same as before (PHY-3). Extra valves could be used to mitigate the effects of an integrity type of attack, if the malware does not manipulate them. The working of the pumps was only important to overflow or empty all drums (PHY-5). The state of the pumps would not have to be altered to overflow or empty one drum; as long as the valves are manipulated correctly.

[a] [c] [e] At first, the malware did not manipulate additional actuators when they were introduced. [b] [d] [f] When the malware uses an analyzer, it will notice which pumps and valves are manipulated by the HMI.

When we implemented a safety system we were not able to put the environment in a critical state (i.e., overflowing or completely empty) any more (PHY-7). [g] The safety system prevented the environment from reaching a critical state. We were still able to infect the system and attack the availability of the HMI.

We added a process to the physical environment that mixes two different liquids into a mixed liquid with motors (PHY-8). The malware was still able to overflow or empty all drums. [h] The malware was also able to manipulate the motors and second liquid flow when the analyzer was used. This meant that if multiple processes are controlled in the same environment, it would be possible to only attack one process.

The obtained results are analyzed in the next chapter.

| | Test 1 | | | Test 2 | | | Test 3 |
|---|---|---|---|---|---|---|---|
| | Infect | Impact availability | Impact integrity | Infect | Impact availability | Impact integrity | Impact integrity |
| PHY-1 | ✓ | ✓ | ✓[a] | ✓ | ✓ | ✓[b] | ✓[b] |
| PHY-2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| PHY-3 | ✓ | ✓ | ✓[c] | ✓ | ✓ | ✓[d] | ✓[d] |
| PHY-4 | ✓ | ✓ | ✓ | | | | |
| PHY-5 | ✓ | ✓ | ✓[e] | ✓ | ✓ | ✓[e] | ✓[f] |
| PHY-6 | ✓ | ✓ | ✓ | | | | |
| PHY-7 | ✓ | ✓ | ✗[g] | ✓ | ✓ | ✗[g] | |
| PHY-8 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓[h] |

If the malware was able to infect a host a '✓' character is noted, otherwise a '✗'. The '✓' character was used in the impact phase if the malware was able to impact the availability or if it could manipulate at least one actuator, otherwise a '✗' is noted. Notes are explained in the text.

Table 4.5: Result from changes in the physical processes.

# Chapter 5 Analysis

This chapter describes the analysis of the results. First, the chapter defines the methodology to analyze the results. Then the chapter states the analysis of the environmental changes. The chapter ends with an answer to the research sub-questions.

## 5.1 Analysis methodology

We analyze the results of both research sub-questions separately.
First, we analyze the result of each environmental change with regard to infection. If the malware's impact was reduced or diminished during the environmental changes, a closer look was taken at why the environmental change reduced or diminished the malware's impact. In some cases, a design decision prevented the malware from adapting to the changed environment. In those cases the design of the malware was revised without having to introduce system knowledge. Furthermore, the lessons learned in the process of building and adapting the malware, were covered with regard to infection. Experiences and difficulties were highlighted and a few environmental changes will be discussed and generalized. Finally, the analyzed results of the environmental changes and the lessons learned will help to answer to the research sub-question: "What system knowledge is required for malware to infect Industrial Control Systems?".

The same steps were followed to analyze the results with regard to the impact on security. Finally, an answer was given to the research sub-question: "What system knowledge is required for malware to impact the security of an Industrial Control System?".

## 5.2 System knowledge needed to infect ICSs

We defined the way malware spreads as the repeated process of infection, privilege escalation and propagation in Section 1.3. The focus lies on infection and propagation because the used exploits to infect the ICS do not require administrator privileges.

### 5.2.1 Analysis of environmental changes

Tables 5.1 and 5.2 were obtained from analyzing the obtained results and describe the required knowledge for the malware to spread to the ICS based on each environmental change.

Section 4.3 showed that the infection-phase was not affected by changes in the PLC configuration or physical environment. The explanation would obviously be: when the malware spread, it did not have a direct connection to the PLC or physical environment until it reached the HMI. Therefore, changes in the PLC or physical environment would most likely not influence the infection stage.

### 5.2.2 Lessons learned

Lessons were learned during the development of the malware, when the impact of the malware on the unchanged environment was determined and when the effect of the environmental changes were tested.

| Change | HMI system knowledge needed to infect the ICS |
|---|---|
| HMI-1 | During this environmental change the firewall was enabled. The firewall blocked one of the malware's scanners. This way the malware could not find or infect the HMI. |
| HMI-2 | The HMI's Operating System was changed from Windows XP to Windows 7 SP0 and Windows 7 SP1. One vulnerability that was used by the malware, was patched in higher versions of the OS. If an exploit does not work on a version of the OS, then another exploit is needed or the knowledge that the OS version is not present in the ICS. |
| HMI-3 | The IP address of the machine was changed. No system-knowledge about the IP address was needed since the malware determined the IP address of the HMI with a scanner. |
| HMI-4 | Data Execution Prevention (DEP) was enabled, but the malware was not influenced by DEP. In the general case, this could depend on the used vulnerability and exploit. |
| HMI-5 | Address Space Layout Randomization (ASLR) was enabled, but the malware was not influenced by ASLR. In the general case, this could depend on the exploited software and the OS version. Windows OSs prior to Vista did not support ASLR and not all software was compiled to support ASLR. If the software and the OS support ASLR then it could be harder to exploit a vulnerability. |
| HMI-6 | The autoplay settings were changed. This environmental change was specifically included for one of the exploits used by the malware. The user had to navigate to the infected folder when autoplay was disabled. |
| HMI-7 | The paths used by the HMI were changed. No knowledge about the project paths used by the HMI was needed. |
| HMI-8 | The HMI's update intervals were changed, but no system knowledge was needed about the HMI update intervals. |
| HMI-9 | The HMI machine was logged in as an unprivileged user. The rights of a user could be determined at runtime. Not all payloads required administrator rights. No administrator privileges were needed to spread to the ICS. |
| HMI-10<br>HMI-11 | The environment was changed so that multiple HMIs were supervising the ICS. No system knowledge about the number of HMIs was needed. The malware should take multiple HMIs into account. |
| HMI-12 | The environment was setup such that two HMIs supervised two PLCs each. No system knowledge about multiple HMIs that supervise multiple different PLCs was needed. |
| HMI-13 | An attempt was made to add a backup connection between the HMI and PLC. This was not possible and therefore the results of this environmental change are unknown. |
| HMI-14 | The program winpcap was deïnstalled from the system. No system knowledge about winpcap or installation of winpcap was needed. |
| HMI-15 | The HMI was isolated from the Internet. Due to the testbed's topology it effectively air-gapped the system. The attacker needs to know if the ICS is connected to the outside. Remote exploits can not be used to infect an air-gapped ICS. This means an attacker should prepare other methods for getting past the air-gap. |

Table 5.1: Required system knowledge about the HMI.

First of all, a general notice. Developing malicious software felt largely the same as developing regular software with C(++). The only difference was the *intention* with which the program was written: it was written to infect machines by exploiting vulnerabilities and executing a payload on the target machine.

Spreading to another system largely depended on vulnerability that was exploited and the exploits that were used. The vulnerability affected the chance and the system knowledge that was required to spread to a system. For instance, if an exploitable vulnerability was found that existed for a long time and was not patched, it could be possible to create an exploit that worked on all currently used versions of the software on all different (Windows) operating systems. This way, no knowledge about the software version or operating system version is needed.

| Change | Network system knowledge needed to infect the ICS |
|---|---|
| NET-1 | Simulated traffic was added. No system knowledge was needed about other network traffic to infect the ICS environment. |
| NET-2 | An instable network connection was simulated. No system knowledge was needed about network loads. |
| NET-3 | The environment was changed so that legitimate traffic was added. The attacker should assume other legitimate traffic is present. |
| NET-4 | Previously captured traffic was replayed, but this did not affect the malware. |
| NET-5 | The control system was put in another VLAN. The same analysis as in HMI-15 was applicable. |

Table 5.2: Required system knowledge about the network.

When an exploit is used that exploits a vulnerability in an application, knowledge is needed about:

- The application that is exploited: Does the targeted ICS use the vulnerable application? The targeted application should be installed (and running) on the target machine(s). Some ICS software vendors note customers on their site. This can be used to determine if the target uses software from a specific vendor. If a target is noted as a customer by a certain ICS software vendor, he will most likely use their software;
- The application version: Not all versions of the application might be vulnerable. Older versions might not contain the exploitable code/service while newer versions might be patched. This depends on the vulnerability and exploit;
- A vulnerability: The vulnerability in the application is exploited by the exploit. Exploits can be obtained in several ways. The first way would be to find a vulnerability and write an exploit for it. Some ICS software is available to download and use [1] [2]. Another possibility would be to use a 'known vulnerability' with an available proof of concept exploit. The last way of gaining an exploit would be to buy it. Some companies [3] [4] [5] sell ICS exploits as part of their business model; and
- The Operating System version: Exploits do not always work on all Operating System versions. Newer OS versions might provide better security by improving or providing new security features. The exploit can work on different OS versions depending on the vulnerability and the exploit.

When a service of the OS is targeted, knowledge is needed about:

- The Operating System: Exploits written for a specific OS will not work on a completely different OS. For instance; if a Windows service is exploited it is unlikely that the same service can be exploited in the same way on a Linux machine;
- The Operating System version: Not all versions of the service might be vulnerable. Patches and service packs can include updates for services; and
- The configuration: Some settings can influence the way a service works and in some cases how and if an exploit works.

When malware is used in a targeted attack it needs to know when it has reached its target. If the malware has not reached its target yet it should spread further. Otherwise, if the malware has reached the target, it should use the target-specific payload. To determine if the malware has reached its target, it should verify if an attribute/characteristic is present. This requires system knowledge about the target since an attribute/characteristic should

[1] http://automation.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/simatic-wincc/
[2] http://igss.schneider-electric.com/
[3] http://gleg.net/agora_scada.shtml
[4] http://revuln.com/
[5] http://www.vupen.com/

be known in advance. Examples of characteristics or unique properties are: a MAC address; an IP address (or range); the presence of a specific software package; a Windows license key; the use of industrial protocols and/or a domain name or computer name. The malware will execute its payload on the systems that have the characteristic or unique property. If a non-unique property is used, then the malware can possibly impact nontargets.

### 5.2.3 Conclusion

The analysis and the learned lessons were used to list the knowledge needed to infect an ICS. First, the knowledge needed during the development is discussed and then the knowledge needed that should be obtained at runtime. The system knowledge needed during development to infect an ICS is listed here.

The interesting findings were that the malware developer needs to know:

- **If custom firewall rules are configured and which custom firewall rules are configured**.
  If custom firewall rules were configured, then the firewall can block the malware's scans or network communication. If the firewall blocks a scan, then the malware will not find any other machine which can result in the malware not executing a remote exploit against connected and vulnerable targets.
- **If the ICS is physically isolated from other networks (air-gapped)**.
  If the ICS is air-gapped, then the malware will need to be moved through the air-gap. This implies that at least one scenario and exploit should be developed that is compatible with physically moving the malware through the air-gap. A few example scenarios for moving the malware through the air-gap are by physically transporting, walking, dropping or throwing the malware on a USB stick, laptop or WiFi-hotspot device through the air-gap. After the malware has reached the ICS it should be executed. This can require an exploit.
- **A unique property of the target to determine if the malware has reached its target**.
  Since the malware should behave differently when it has not reached its target than when it has reached its target, it should be able to make that distinction.

And the malware developers that target ICSs also need to know:

- **The Operating System (e.g., Windows, Linux) and OS versions used in the ICS**.
  (Malicious) software written for Windows OSs will generally not work on Linux based OSs and vice versa. Knowledge about the version is needed if the used exploits do not run on all versions of a specific OS. If the ICS contains versions of the OS that are not compatible with the exploit, then the malware will not be able to infect the ICS.
- **The software and software versions used in the ICS (to supervise and control the PLCs)**.
  More exploits can be included in the malware depending on the knowledge of the software used by the targeted ICS. If the attacker downloads/buys the software he could search for remote exploits and experience how the software reacts to exploits.
- **Vulnerabilities and exploits**.
  One or preferably more exploits that work with the OS and software versions are needed. If knowledge is obtained about the OS/software versions or patch frequency, a decision can be made if known vulnerabilities can be used. A privilege escalation exploit would be needed if the malware contains functionalities that require administrator privileges.

System knowledge that the malware needs to obtain at runtime:

- **Which machines/devices can be targeted**.
  One way to infect other machines, is to infect machines in the LAN. A scan can be performed to learn which machines are present in the LAN. Other ways to infect other machines are to infect shared media, such as, USB-drives, printers or network shares.

- **If a machine is vulnerable to the remote exploits contained by the malware**.
  If a machine is vulnerable, then the malware will run the corresponding exploit and it will propagate to the machine.
- **If the current machine is the final target machine**.
  The malware needs to determine if the current machine is the final target. If not, the malware will try to spread further. If the current machine is the final target, then the malware will run a special payload. It should check if a unique property, that was defined during development, is present in the current machine or its environment.

## 5.3 System knowledge needed to impact the security of ICSs

The impact on security was defined as an attack on availability or integrity. An attack on integrity was performed when the malware changed at least one of the actuators of the system. An attack on availability was successful when it stopped engineers from supervising or controlling the ICS.

### 5.3.1 Analysis of environmental changes

Tables 5.3, 5.4 and 5.5 were obtained from analyzing all obtained results. The tables indicate for each environmental change the required knowledge to impact the security of the ICS.

The payloads used to impact the ICS were not affected by network changes, as seen in Section 4.3.

### 5.3.2 Lessons learned

Payload PAY-8, which targeted availability, succeeded almost every time during the environmental changes because DoS attacks on the HMI required little system knowledge. A few examples of attacks that targeted the availability of the HMI were; poisoning the ARP cache, terminating the HMI process or simply shutting down/rebooting the HMI machine. Those availability attacks were less dependent on influences from the PLCs or physical environment than integrity attacks.

Spoofing traffic turned out to be a big challenge. As mentioned in Section 3.4.3 Windows XP SP2 and newer place restrictions on the creation of raw TCP sockets. Without raw sockets it is not possible to modify Ethernet headers like: destination MAC address; source MAC address; or source IP address. This made spoofing packets a lot harder. One way to create raw TCP sockets was to use winpcap. Winpcap is a network interface and provides the options to send raw data, but it has to be installed first. This enables attackers to spoof packets with malware.

When the MITM functionality was developed, the malware needed to listen and respond to requests. The malware listened to what was send over the network. It had to interpret Modbus traffic from connected PLCs and filter out all the other traffic (e.g., HMI synchronisation traffic or HTTP traffic). The malware used a filter that only let traffic to and from the connected PLCs through. A command-line tool, netstat [6], was used to enumerate the connected PLCs (i.e., ports and IP addresses). This information was also used to differentiate between different PLCs. To select the right connections with netstat, the name of the service responsible for communicating with the PLCs was required.

Functionality was needed to analyze the physical environment when no prior system knowledge was available. The analyzer first enumerated all connected PLCs and allocated enough memory. It then listened to the traffic for a predefined number of useful responses (packets). The analyzer kept track of all minimum and maximum values that the HMI read (read single or multiple registers) from the PLC. It also kept a record of all register-addresses the HMI wrote to (write single or multiple register). Those addresses were used by the actuators and would later be used to write to (integrity attack). When the analyzer had almost received the predefined number of useful responses, it would also record read requests so they could later be used to spoof traffic.

[6] in environmental change PLC-7 test 2

| Change | HMI system knowledge needed to impact the system |
|---|---|
| HMI-1 | During this environmental change the firewall was enabled. No system knowledge about the firewall configuration was needed to impact the system. It was possible to create a local DoS attack or spoof packets to take over a connect. |
| HMI-2 | The HMI's Operating System was changed from Windows XP to Windows 7 SP0 and Windows 7 SP1. No system knowledge about the used OS version was needed during the environmental tests. |
| HMI-3 | The IP address of the machine was changed. No system-knowledge about IP addresses of the HMI was needed since the malware was already on the machine and was able to request the IP addresses of each network interface at runtime. |
| HMI-4 | Data Execution Prevention (DEP) was enabled, but this was not relevant at the stage where the malware had already infected the system. |
| HMI-5 | Address Space Layout Randomization (ASLR) was enabled, but this was not relevant at the stage where the malware had already infected the system. |
| HMI-6 | The autoplay settings were changed. No knowledge about autoplay was needed. |
| HMI-7 | The paths used by the HMI were changed. No knowledge about the project paths used by the HMI was needed. |
| HMI-8 | The HMI's update intervals were changed, but this did not influence the malware. The used physical environment did not contain time-critical processes. The outcome of this environmental change could be different in more time constraint or real-time environments. |
| HMI-9 | The HMI machine was logged in as an unprivileged user. The malware also had limited privileges. The rights of a user could be determined at runtime. Not all payloads required administrator rights. The malware did not need administrator privileges to impact the ICS. |
| HMI-10<br>HMI-11 | If multiple HMIs were present and supervising the same (part of the) system the malware should perform a MITM attack on as many HMIs as possible. For instance, if multiple HMIs supervise the same (part of the) system and the malware performs a MITM attack on all except one, then that HMI would be capable of reading the true system state and possibly be used to intervene. No system knowledge was needed to impact the security. The number of HMIs only affected the success of the MITM attacks. To perform a successful MITM attack, the malware needs: knowledge about the HMIs that supervise the process; an exploit to reprogram the PLC like Stuxnet did; or a thorough scan and be able to communicate with all HMIs to perform a MITM. |
| HMI-12 | The environment was setup such that two HMIs supervised two PLCs each. The same knowledge was needed as in environmental changes HMI-11 and PLC-7 combined. |
| HMI-13 | An attempt was made to add a backup connection between the HMI and PLC. This was not possible and therefore the results of this environmental change are unknown. |
| HMI-14 | The program winpcap was deïnstalled from the system. Winpcap was not needed to impact the availability or integrity of the ICS but the malware could not perform an MITM attack without it. |
| HMI-15 | The HMI was isolated from the Internet. Due to the testbed's topology it effectively air-gapped the system. No knowledge was needed about whether the ICS was air-gapped or not, to impact the ICS. Once the malware infected the air-gapped ICS it could proceed as usual. The malware could probably not create a backdoor to communicate to a command and control server. |

Table 5.3: Required system knowledge about the HMI.

When the analyzer was done, the malware started the MITM attack. It disconnected the PLCs from the HMI and wrote randomly either the lowest or highest value to each writable register address. It assumed that the lowest or highest values were most likely the worst settings, to create a potential dangerous state. Although not implemented, it would be possible to analyze the effect of the written state and if the current state did not cause the appropriate effect write a new random state.

Other, but more protocol specific, ways exist to impact the security of the ICS. The Modbus/TCP protocol specifies

| Change | PLC system knowledge needed to impact the system |
|---|---|
| PLC-1 | The addresses that were used for the sensors and actuators were changed. The malware could analyze which addresses were used at runtime if the PLC was connected via Ethernet and no encryption was used. |
| PLC-2 | The IP address of the PLC was changed. No system-knowledge about IP addresses was needed since the IP addresses of the PLCs were determined at runtime. |
| PLC-3 | The PLC initially used the default Modbus port. Another port was configured for the PLC to listen to. The ports used by the PLCs were determined at runtime. |
| PLC-4 | The PLC's response latency was changed but this did not reduce the malware's impact. |
| PLC-5 | A serial connection was setup for the HMI to communicate with the PLC. If a serial interface was used to connect the PLC to the HMI then the malware would need to know the baud rate. When Modbus was used then the transition mode (i.e., RTU or ASCII) would also be needed. It was not possible to analyze the traffic because it was not possible to open the same port twice without modifying drivers or hardware. This means knowledge about the environment and which addresses were used to write to was needed. |
| PLC-6 | The protocol used by the HMI and PLC was not changed to DF-1 because it would not create any new insights compared with the previous environmental change. |
| PLC-7 | The number of PLCs was changed but did not stop the malware from impacting the security during the environmental changes. |

Table 5.4: Required system knowledge about the PLC.

| Change | Physical environment knowledge needed to impact the system |
|---|---|
| PHY-1 | The number of pumps in the environment was changed. Some actuators are be more important than other actuators. |
| PHY-2 | The number of drums were changed. The malware did not need knowledge about the amount of drums. |
| PHY-3 | Addition in- or out-valves were added. The malware did not need to know the exact number of in- or out-valves to impact the environment. Additional valves could be used to prevent the system from reaching a critical state. |
| PHY-4 | The amount of liquid a drum could hold was altered. The malware did not need to know about this. It could have been used to check if the integrity attack put the system in a critical state. |
| PHY-5 | The working of the pumps was changed and this influenced the malware when it used the hardcoded values. |
| PHY-6 | The flow speed of the liquid was changed, but this was not important for the malware to impact the system. |
| PHY-7 | When a safety system was added, it prevented the malware from overflowing or emptying drums. The attacker needs to know if a safety system is in place. This changes the available options to impact the integrity of the system. |
| PHY-8 | A process that required engaging motors for mixing the fluids in the tanks was added. When the malware targeted one process, it did not require knowledge about other processes. |

Table 5.5: Required system knowledge about the physical processes.

the possibility to set/force the PLC to a 'listen only'-mode. This creates a DoS attack since the HMI would not be able to read or write values from or to the PLC anymore. Another option in the Modbus/TCP protocol would be the 'restart communications'-option which forces a restart and power up self-tests of the PLC. If this is performed repeatedly, it will be impossible for the HMI to control the system. Yet another option would be the following scenario: every time a HMI sends a request, the PLC should respond within a timeframe. The PLC will usually be able to reply under normal circumstances, otherwise it has the option to send a 'Slave Device Busy Exception

Code Delay'-response which notifies the HMI that the PLC is busy. An attacker can use this functionality to postpone an action or an alarm by responding with a 'Slave Device Busy Exception Code Delay' to a request. This would also create a DoS.

### 5.3.3 Conclusion

The analysis and the learned lessons were used to list the knowledge needed to impact the security of an ICS. The system knowledge needed during development to impact the security of an ICS with malware is described here.

An interesting finding was that a malware developer would need to know:

- **If it is possible to create a backdoor that has access to the Internet.**
  A backdoor, created by the malware, that is able to communicate with a command and control server creates, the possibility to analyze the environment manually. The attacker can use the backdoor to determine which software is used to supervise the plant and which communication interfaces are used to communicate with the PLC. The attacker can also obtain records of the traffic and HMI configurations and (project) files. The HMI project files contain detailed information about the environment. A record of the traffic can reveal the communication protocols that are used, information about the processes, and how the actuators are controlled. An optimized payload can be developed based on this analysis;

A malware developer that target ICSs also needs to know:

- **The software used to supervise and control the PLCs or the software used to program the PLCs.**
  If the attacker knows which HMI software and which version of the software is used, it would be possible to easily enumerate all connections made by the HMI and thus enumerate all connected PLCs. If the software is obtained then the attacker can experience how the software responds to MITM and DoS attacks. If the developer knows which application is used to program the PLCs he might use it to exploit the application to reprogram the PLC;
- **The communication interface and protocols used in the ICS, especially for communicating between the HMI and PLC.**
  If the attacker wants to impact the integrity of the system but does not reprogram the PLC, then the attacker would have to communicate to the PLCs. The communication to the PLC is needed to manipulate the actuators and attack the integrity of the processes in the ICS. This would require knowledge about the communication interface and protocols.

  It is possible to support multiple (most commonly used) protocols and determine the used protocol at runtime. If a protocol is used that uses authentication / encryption it will become harder or impossible to analyze the environment. The malware can still impact the integrity of the environment if it writes random data to (all) addresses or if it uses another way to find the actuators. The use of authentication prevents spoofing traffic and performing MITM attacks. This would make it harder to completely hide the malware's presence.

  Depending on the protocol, one or more interfaces (e.g., Ethernet, serial port) are supported. As seen in the environmental changes, the used communication interface matters. The malware could analyze the traffic and perform MITM attacks when Modbus/TCP was used in the ICS. The malware could not analyze the traffic or perform a MITM attack any more when Modbus over a serial connection was used. When a serial connected was used, it enabled an extra attack on availability by stopping the process that uses the serial port and connecting and occupying the port. If it is not known which interface is used then the malware will have to support both Ethernet and serial port interfaces. If a serial connection is used, then knowledge is needed about: the used baud rate; the number of data- and stop-bits; and depending on the protocol, which mode is used;
- **Knowledge about the physical processes controlled by the ICS.**
  Since the attacker is performing a targeted attack, he should already know what the ICS does. The attacker should delve deeper into what processes are controlled by the ICS if he wants to put the system in a critical state. Knowledge about possible time-critical processes can create more options for the malware to impact the security of the system;

- **What equipment is in place and how the processes are controlled**.
  The malware should manipulate the actuators if it wants to impact the integrity of the ICS. If the malware wants to manipulate the processes in a controlled way, it should know what actions the actuators take depending on the values. This requires knowledge about the installed sensors and actuators; and
- **If a safety system is in place and what it protects**.
  A safety system can protect the processes from reaching undesired or critical states. It can prevent malware from putting the processes in a critical state.

System knowledge that the malware needs to obtain at runtime:

- **A list of connected PLCs, their IP addresses and ports**.
  During the attack, the malware needs to know which PLCs are connected. If the malware can not figure out which PLCs are connected it will likely not be able to impact the integrity of the ICS. If a malware developer has detailed information on the processes and how they are controlled, it is possible to hard-code or search for this information at runtime. Otherwise, the malware can still impact the integrity of the ICS by manipulating every actuator the malware can find.
- **Which addresses are read and which addresses are written to by the HMI**.
  The malware analyzes the environment to know which addresses are read and which addresses are written to. The values that are read from the PLC can be recorded to spoof traffic. The HMI writes to the addresses that are used by the actuators. The malware can use those addresses to manipulate actuators and put the system into a critical state (e.g., overflowing). If the malware analyzes the minimum and maximum values read from the sensors it is possible to determine if the current manipulated state affects the state of the system.

# Chapter 6 Related research

This chapter will cover the related work discovered during the literature study phase.

## 6.1 State of the art in ICS simulation

Several testbeds were found during the literature study, such as the one described by Genge et al. [13] that was based on Emulab. Emulab is a network testbed that consists of hundreds of PCs hosted by the University of Utah. Emulab can be used to create cyber components (e.g., SCADA servers, corporate network) and the paper described a setup where Matlab Simulink was used to create physical processes. The testbed was used to measure the impact of attacks against the cyber and physical parts of the system.

Another testbed described by Fovino et al. [10] uses an experimental test-bed, deployed in Italy. It is used to simulate attack scenarios against SCADA systems and to test countermeasures in a safe way. The environment reproduces the dynamics of a physical power plant with real PLCs and sensors. Four attack scenarios were presented: a DoS attack on the Remote Authentication Dial in User Service (RADIUS); a computer worm; a process network malware infection; and phishing attacks together with local DNS poisoning. A set of countermeasures are proposed, such as the use of encryption and authentication for regular TCP/IP traffic as for SCADA traffic, and filtering and monitoring with firewalls and IDSs. This research also showed that the four attack scenarios were realistic and feasible. The process network malware infection attack was similar to the malware developed in this thesis.

Genge et al. determined the impact of network infrastructure parameters to the effectiveness of cyber attacks against ICSs [12]. The research determined if network delays, packet losses, background traffic and control logic scheduling time influenced the outcome attacks against networked ICSs. The attack scenario involved an attacker that is able to send legitimate Modbus packets/commands with the goal of bringing the control system in a critical state. The research showed that network delays, packet losses and background traffic have limited effect on the attack. This was also true for the environmental changes that were performed in this thesis.

Chunlei et al. assessed a representative simulated environment for SCADA security analysis [5]. The environment included several PLCs from different vendors. Their results demonstrate the effectiveness and practicability of the simulation environment by showing that it was possible to perform an attack scenario. The attack scenario consisted of: 1) gaining access to the simulated SCADA system; 2) Gaining local SCADA access via the enterprise network and scanning the network; and 3) Compromising the master device by exploiting a vulnerability. The malware created during this thesis was also able to perform these steps on our environment.

Perez-Garcia et al. determined how different design choices, e.g., use of specific tools, can affect the repeatability of the experiments of an emulation testbed [22]. The testbed was based on the Emulab software. They showed that Emulab's scheduling mechanism, built-in packet generators and Iperf can sufficiently support repeatable experiments. They also showed that TCPreplay did not support repeatable experiments and that an alternative tool named TCPivo should be used instead. TCPreplay was used in this thesis for environmental test NET-4. Repeatable experiments were not required for the purpose of replaying traffic to see if it interfered with the malware.

Reaves et al. propose a virtual testbed for industrial security research [23] that was created using Python. The virtual devices were able to communicate with each other, the simulator, and with actual ICS devices. The virtual devices supported multiple protocols. Logging devices were developed to capture and replay virtual system traffic.

Genge et al. analyzed cyber-physical attacks on networked Industrial Control System [11]. This paper defined an efficient approach for conducting experiments that measure the impact of attacks on the cyber and physical

components of the ICS. It represented an advancement over existing approaches by using a hybrid architecture. It emulated the protocols and components such as the SCADA servers and PLCs and simulated the physical layer using Matlab Simulink.

Morris et al. describes the Mississippi State University SCADA security laboratory [20]. The laboratory contains multiple power and energy ICSs (i.e., a raised water tower, gas pipeline, factory conveyor and industrial blower) with functional physical processes controlled by commercial hardware and software. They made a distinction between two categories based upon the communication type; serial port or Ethernet communications. The malware developed in this thesis was also able to infect and impact both categories.

Wu and Kubara made a comparison of the tools and simulators for control system security studies [30]. They classified the tools/simulators into four types according to their important characteristics. The four types were: single simulation tools for devices; network simulation tools; integration simulation tools; and realistic testbed tools. Finally they compared the complexity, realistic capability real-time simulation performance and simulated security events. The environment used in this thesis can be categorized as an integration simulation tool since it is setup by integrating different kinds of security simulation on devices, data, and network. The comparison states that the integration simulator has a 'middle complexity', a 'high realistic capability' and a 'low real-time simulation performance'. These statements seem representative for our environment because it was relatively easy to setup the environment, real SCADA software was used and this setup would likely not be able to simulate large ICS environments in realtime.

## 6.2 State of the art in malware development

The possibility to create ICS tailored malware with simulation software (MAlSim) has been researched [3, 21] by Carcano and Fovino et al. Both papers describe the same research that was performed by the same researchers but they papers were published in different journals. These papers were the most relevant papers for this thesis. The papers describe the development of a worm that is able to communicate with PLCs over Modbus/TCP. The worm has two infection steps. The worm first infects PCs in the Company Intranet from the Internet. Then, if a infected PC opens a VPN connection, the worm spreads itself through the VPN and infect PCs in the process network. If the worm discovers Modbus slaves in the network, it will communicate with them. Two scenarios were created, a scenario to perform a DoS attack and a scenario to take control of the slaves. The DoS attack was performed by sending Modbus packets to desynchronize or completely interrupt the master/slave Modbus command flow. They found that systems with a low scanning rate desynchronized faster than systems with a high scanning rate when they were under attack. The second scenario is created with the intention to take control of the slaves by taking advantage of the lack of authentication and integrity verification. The worm code was a lot more complex, but, potentially, also a lot more dangerous. The worm was developed in a incremental manner and three prototypes were made in the process. The first prototype forced each coil in a sequence of coils to either ON or OFF. Closing or opening the coils could have a very high impact in a SCADA system. The second prototype targeted the input register. It wrote the biggest allowed value (i.e., a 16-bit word) to all registers. The final prototype read the state of the coils and wrote the inverted state to each coil. This attack strategy completely changed the coil configuration. It was stated that the attacker has to know at least the high-level details of the architecture of the system under attack in order to write an "attack strategy module" with high effectiveness. Which high-level details of the architecture of the system were needed was not clearly defined. This thesis's goal was to describe the needed knowledge, including details of the architecture, needed to write malware with payloads ("attack strategy modules").

## 6.3 Comparable research with regard to impact on ICSs

Several ICS attack models and scenarios have been defined and the attack scenarios were staged at a testbed [29]. An experimental setup with four water tanks was controlled over a wireless network. The setup was used to illustrate attack scenarios, their consequences, and potential counter-measures. A figure in the paper describes a three dimensional cyber-physical attack space which depicts several attack scenarios that were described in the paper. The figure used in the paper is shown in Figure 6.1. One of axis describes the 'a priori' system knowledge

needed. According to the picture, DoS and replay attacks require very little to no system knowledge while

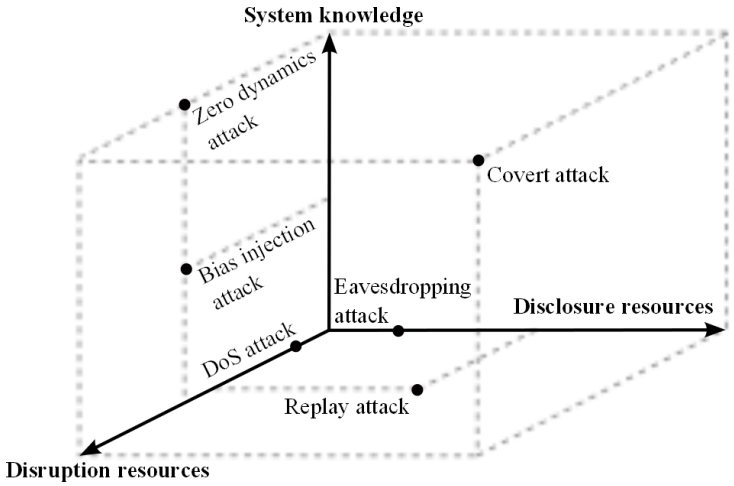



Figure 6.1: The cyber-physical attack space. [29]

integrity attacks require at least a little system knowledge. This is in line with the findings in this thesis.

Cárdenas et al. show how they are able to detect attacks that change the behavior of the targeted system by incorporating knowledge of the physical system [4]. They found that the most effective attacks were max/min attacks (i.e., where maximum or minimum values are used for sensors/actuators). However not all max/min attacks where able to put the system in a critical state. During an integrity attack it was possible to get the system in an unsafe state but it took 20 hours before the unsafe state was reached. They stated that for physical processes with slow-dynamics it would be possible for operators to detect the attack and take proper actions. They also found that, in general, DoS attacks do not put the plant in an unsafe state. In this thesis we also used max/min attacks when the malware used an analyzer. After the malware analyzed the environment, a max/min attack was performed.

Stealthy deception attacks were also investigated [2] where the attack was implemented at a physical canal. They present a linearized shallow water Partial Differential Equation (PDE) system that can model water flow in a network of canal pools. They use this PDE system to withdraw or steal water from the pools. The detectability of such attacks is briefly discussed. The proposed attack is tested in simulation and in practice on the Gignac canal in Southern France. The field experiment showed that the attack could steal water stealthily from the canal until the end of the attack.

Slay et al. discusses the lessons learned from the Maroochy water breach [27] where an attacker used a laptop and a radio transmitter to take control of 150 sewage pumping stations right after the system was installed. Over a three-month period the attacker released one million liters of untreated sewage into a stormwater drain from where it flowed to local waterways. The faults that occurred included: unexplained pump station alarms; increased radio traffic that caused communication failures; modified configurations for pump station software; pumps running continually or turned off unexpectedly; and computer communication lockups and no alarm monitoring. In the end, an engineer who was monitoring every signal passing through the system, discovered that someone was deliberately causing the problems. This case has been cited around the world as an example of the damage that could occur if SCADA systems are not secured.

Chapter 7

# Conclusions and future work

In the introduction we identified the following problem, namely, that malicious software (malware) is omnipresent and that malware is now able to reach Industrial Control Systems (ICSs). This could cause financial loss or physical damage. Developing malware that can infect and impact ICSs requires a certain amount of prior system knowledge. If the information needed to develop malware for a specific target was kept secret by ICS managers/employees, it would be harder to target that ICS using malware. This brought up the question: "What system knowledge is needed for a malware developer to create malware to infect and impact an Industrial Control System?". We first need to answer the following two sub-questions: "What system knowledge is required for malware to infect Industrial Control Systems?" and "What system knowledge is required for malware to impact the security of an Industrial Control System?". The methodology for answering the sub-questions was: first, setting up an environment and developing malware; then we prepared a list of possible changes in the environment after which the environment was changed according to one of the changes in the list; finally, the changes that reduced or diminished the effects of the malware were determined.

This methodology could not be executed without learning about Industrial Control Systems first. The Purdue Model for Control Hierarchy described a model for typical control systems and the relevant components. The difference between ICSs and regular Information Technology (IT) systems was explained and the described components were used to setup an environment. We set up an environment to represent a chemical plant which contained drums, pumps and valves. We developed malware that was able to infect the plant and impact the integrity and availability by disrupting plant supervision and overflowing or emptying the drums.

We prepared a list of possible environmental changes and the list was reviewed and completed by ICS and malware specialists. This resulted in a list of 35 environmental changes. We changed the environment according to an item on the list and determined if the change reduced or diminished the effects of the malware. System knowledge was needed if the malware was unable to infect or impact the security of the plant when this was not caused by a design decision. These findings were analyzed and together with the learned lessons the sub-questions were answered.

This chapter concludes the research. First, the main research question will be answered. Then, possible improvements of the research are stated. Furthermore, the impact of the research is discussed. Finally, possible future work is listed.

## 7.1 Answer to the research question

This section covers the answers to both sub-questions and the research question. The research question is stated as follows:

"What system knowledge is needed for a malware developer to create malware to infect and impact an Industrial Control System?"

The research question was divided into two sub-questions which were answered separately. The answers are combined to answer the research question. We expected some answers since they were quite obvious but we also obtained a answers that were quite interesting. A distinction is made between the expected and interesting items.

### 7.1.1 Answer to the first sub-question

We stated the second research question as:

"What system knowledge is required for malware to infect Industrial Control Systems?"

The way malware spreads was defined as the repeated process of infection, privilege escalation and propagation. The focus, to answer the research question, lies on infection and propagation because the used exploits do not require administrator privileges to propagate.

The interesting findings were that the malware developer needs to know:

- If custom firewall rules are configured and which custom firewall rules are configured.
- If the ICS is physically isolated from other networks (air-gapped).
- A unique property of the target to determine if the malware has reached its target.

And the malware developer that target ICSs also need to know:

- The Operating System (e.g., Windows, Linux) and Operating System (OS) versions used in the ICS.
- The software and software versions used in the ICS (to supervise and control the Programmable Logic Controllers (PLCs)).
- Vulnerabilities and exploits.

### 7.1.2 Answer to the second sub-question

We stated the second research question as:

"What system knowledge is required for malware to impact the security of an Industrial Control System?"

The impact on security was defined as an attack on availability or integrity. An attack on integrity was performed when the malware changed at least one of the actuators of the system. An attack on availability was successful when it stopped engineers from supervising or controlling the ICS.

An interesting finding was that a malware developer would need to know:

- If it is possible to create a backdoor that has access to the Internet.

A malware developer that target ICSs also needs to know:

- If a safety system is in place and what it protects.
- The software used to supervise and control the PLCs or the software used to program the PLCs.
- The communication interface and protocols used in the ICS, especially for communicating between the Human Machine Interface (HMI) and PLC.
- Knowledge about the physical processes controlled by the ICS.
- What equipment is in place and how the processes are controlled.

### 7.1.3 Conclusion

Malware developers need to acquire certain knowledge to launch a targeted attack on an ICS.

If an attacker wants to impact the security of the ICS with malware, he needs to infect the ICS first. This requires knowledge about what OSs (e.g., Windows, Linux) need to be infected. One or more exploits compatible with the OS are needed to infect the targeted machines. Knowledge about the OS version is needed, depending on the vulnerabilities that the exploits target. If the target is not connected to the Internet (i.e., completely air-gapped) then an attack scenario with corresponding exploits is needed (e.g., a scenario where the malware infects USB-drives). Knowledge about firewalls and their rules will enable the attacker to develop malware that can spread

and scan through networks without being blocked. The attacker should know a (unique) property of the target so the malware can use that to detect if it has reached its target.

If the malware reaches a machine that can control the environment it will try to impact the integrity and/or availability of the system. If the malware can create a backdoor that is able to communicate with a command and control server, then it becomes possible for the attacker to analyze the environment manually. Otherwise, more knowledge would be needed during development, such as knowledge about the physical processes that are controlled by the ICS and how the processes are controlled. Knowledge about the (Supervisory Control and Data Acquisition (SCADA) or HMI) software used at the ICS can be used to develop extra exploits and enumerate connected PLCs at runtime.

## 7.2 Improvements

During the research some improvements came to mind. Ideally, this research would have been done by two persons. One person would create and operate the ICS and the other would develop the malware. If both persons would not discuss their setup and code, then it would be possible to more accurately measure the knowledge needed to develop malware. The malware developer can test his malware on the unknown environment. If he needs more knowledge to make the malware function, he can ask the ICS creator a question about the environment. The ICS creator would provide an answer and thus leak system knowledge. If the question and answers are documented, it would be possible to determine which system knowledge needed to be leaked to launch a targeted attack.

A weak point of the methodology was the generalization. The results of the environmental changes where generalized. The generalizations are made for cases where malware wants to spread using remote exploits and an exploit compatible with portable media. The infected portable media can be used to infect other machines and to get past an airgap. In our case a remote exploit was used to infect the HMI. The answers obtained from the research gave few interesting findings. Nothing spectacular or ground breaking was discovered. The research could have given more interesting findings if a broader question was researched, for instance, “what knowledge is needed to infect and impact the ICS stealthily?”.

If more time would have been spend in creating the questionnaire, then it would have been possible to get more or higher quality answers. If the questionnaire was also send to an ICS or malware community, even more diverse environmental changes could have been obtained. This would also imply that more time was needed to test and analyze the additional information. A small improvement would have been to include privilege escalation exploits to the infection phase. Finally, the environment could have been setup more accurate by including a Demilitarized zone (DMZ). This would add another layer of security. But since the machines and software in the DMZ can contain vulnerabilities as well, it would only require one additional exploit for the malware to get past the DMZ. If the software used in the DMZ is made by the same vendor as the software used in the Operational Technology (OT) network, it could be the case that the vulnerability is present in both instances. This would mean only one exploit is needed to infect both machines.

## 7.3 Impact of the research

The problem description stated that malware was omnipresent and that it would require prior system-knowledge to develop malware to infect and impact the security of an ICSs. The main impact of the research is that it provides a list of system knowledge that is needed to develop malware to infect and impact the security of an ICS. The list is not very large and some items can be guessed or easily obtained. This underlines the importance of securing ICSs by airgapping, updating and patching them. As part of their defence in depth strategy, ICS staff should not share the information described here to make it harder for malware developers to launch targeted attacks.

Attackers that are planning to target an ICS with malware will most likely start by learning as much as possible about their target. One of the easiest ways for attackers (malware developers) to gain Intel about a target is by obtaining the information using social engineering. When ICS management and engineers are aware of the information that should be secret, it would be easier to differentiate between a legitimate question and a social

engineering attack. This way, social engineering attacks can be spotted more easily making it even possible to detect the information gathering phase of a targeted attack.

The result of the research can also be used for forensics and incident response. When a malware sample is obtained, it is possible to more quickly determine if the malware is used for a targeted attack. If most of the information of the checklist is found, it will be likely that it is a targeted attack

## 7.4 Future work

The research also raised questions that could be researched in the future.

It would be interesting to determine the knowledge that is needed to infect and impact an ICS in a stealthy way. The result of the research combined with this research will provide a more complete picture of the required knowledge needed to stealthy infect and impact the security of the ICS. If malware is not stealthy, it can be discovered more easily before the ICS is infected or before it impacted the security of the ICS. If the malware is very stealthy it can take a long time before anyone detects it.

This research can also be performed with an emulated environment with real physical environment parameters or with a real ICS with physical components. This can provide more accurate insight into the knowledge needed to put the system in a critical state. If a real ICS setup is used, it would be interesting to look to the effect of certain protocol specific commands that can be used to control the PLC's behavior. For instance, the Modbus/Transmission Control Protocol (TCP) protocol specifies the possibility to set/force the PLC to a 'listen only' mode. The 'restart communications' option which forces a restart and power up self-tests of the PLC, could be used repeatedly to make it impossible for the HMI to control the system. Future work can include a research to determine the knowledge that is needed to reprogram a PLC. Future research could determine the knowledge needed for malware to communicate with PLCs using different protocols. Does it take more knowledge to use DF-1 instead of Modbus? And how does encryption and authentication exactly influence the developer's required knowledge to create malware to impact the integrity of the ICS? Will authentication between the HMI and PLC completely block the malware from talking to the PLC? Or can the malware recover keys that are used by the HMI once it has infected it? If different HMI software can be obtained, it would be possible to research the behavior of the software on different Man-in-the-Middle (MITM) and availability attacks. This would require HMI software from multiple vendors. Future research could provide more insight into the difference in knowledge that is needed to infect and impact IT systems or Industrial Control Systems.

Future research could determine the knowledge needed for many other different physical environments. Does it take more or different knowledge to impact the security of power plants, than to impact food manufacturing facilities? This could provide ICS managers and engineers a more tailored advice on what information should be kept more securely.

Research could be performed to gain more insights into a human factor in control system security. How can malware manipulate engineers to trick them into making wrong decisions to impact the integrity or availability of the system? Would manipulation of the data shown on the HMI be enough? If malware is able to impact the integrity of the system, it could also do it with the purpose of triggering the safety system instead of putting the system in a critical state. The malware's goal would be to trigger the safety system repeatedly during a MITM attack, to let the ICS engineers suspect that the safety system is not working properly. The malware can try to put the system in a critical state if the engineers take action towards the safety system by, for instance, ignoring it, disabling it or reprogramming it. How would engineers react to such an attack scenario?

Research into the remote access vendors use can provide new attack vectors. Is it possible to use this functionality during attacks? What are the benefits for malware developers to target ICS vendors to obtain details and credentials and attack ICS that way? Does this decrease the knowledge needed to write malware? And how can ICS be secured against these attacks?

Appendix

# Definitions

| | |
|---|---|
| ARP | Address Resolution Protocol |
| ASLR | Address Space Layout Randomization |
| CIA | Confidentiality, Integrity and Availability |
| DCS | Distributed Control Server |
| DCS | Distributed Control System |
| DEP | Data Execution Prevention |
| DLL | Dynamically Linked Library |
| DMZ | Demilitarized zone |
| DoS | Denial of Service |
| GUI | Graphical User Interface |
| HMI | Human Machine Interface |
| ICS | Industrial Control System |
| IDS | Intrusion Detection System |
| IED | Intelligent Electronic Device |
| IGSS | Interactive Graphical SCADA System |
| IP | Internet Protocol |
| IT | Information Technology |
| LAN | Local Area Network |
| MAC | Media Access Control |
| malware | malicious software |
| MITM | Man-in-the-Middle |
| MMI | Man Machine Interface |
| OS | Operating System |
| OT | Operational Technology |
| PID | Proportional-Integral-Derivative |
| PLC | Programmable Logic Controller |
| RTU | Remote Terminal Unit |
| SCADA | Supervisory Control and Data Acquisition |
| TCP | Transmission Control Protocol |
| USB | Universal Serial Bus |
| VM | Virtual Machine |

Appendix

# A Questionnaire

Dear,

A while ago I started writing my master-thesis for my Information Security study at Technical University Eindhoven (The Netherlands). The subject of my master-thesis is: "What system-knowledge is needed for malware to infect and impact Industrial Control Systems?". To research this topic I created malware and set up a simulated ICS environment. I would like to ask a few minutes of your time to fill out this questionnaire.

My methodology was to create malware compatible with an environment and change the environment according to a list of test-cases. This created a lack of knowledge on the malware side. If the malware doesn't work after a change it could indicated the need for more knowledge. Since I know the environment, I'm able to make a biased list and only perform test-cases that wouldn't influence the malware. I can't reliably measure knowledge when the list is biased.

I suppose that I will always be biased, but if you and other people will take a look and add 'environmental change' items to my list, the completeness of the list will increase and the bias from my side will decrease. For this reason, I cannot give you information about my setup or malware. The questions can be found after the test-case-list. Could you please take a look at it? All I need to do next is to perform the test-cases.

Thanks a lot in advance!

## Test-cases; environmental changes

I've started with a list of several possible test-case scenarios. I will test if the created malware is still able to infect and create loss of availability or integrity in a changing environment. The test-cases are categorized:

### Changes on the HMI machine

- Change the OS
- Change IP
- Enable DEP
- Enable ASLR
- Enable firewall
  - and block ping with firewall
- Change project paths used by HMI
- Increase/decrease HMI's update intervals
- Run the HMI as unprivileged user
- When multiple PLC's are used, make the HMI supervise only part of the PLC's
- Add a secondary hot backup server

### Changes in PLC configuration

- Change memory addresses used
- Change IP
- Change port (not 502)
- Change response latency
- Use Modbus over RS232 instead of TCP/IP
- Change protocol to DF-1
- Change number of PLC's
  - and with different protocols

### Change physical processes

- Change number of drums
- Change number of pumps
- Change number of in- or out-valves
- Change amount of liquid drums can hold
- Change working of the pumps

### Change network

- Add simulated traffic

## Constraints

Some constraints I've noticed while I created this list:

1. The setup still resembles the type of plant I started with
2. The PLC simulator I use only supports Modbus over TCP/IP or RS232 and DF-1. To test other protocols, I will need another PLC simulator.
3. Just one Supervisory and Control software package is used. If I would want to use other software, I would need to obtain another exploit. This process will take quite some time while, in my opinion, it doesn't provide much added value.

## Questions

All feedback is welcome:

1. Should any of the test-cases be removed?
2. Would you modify any of the test-cases?

3. Do you have additional test-cases?
4. Is the current list constrained in any other way?
5. Do you have any other remarks I should take into account?

Appendix

# B Questionnaire Response

The following responses were obtained from the questionnaire;

## Constraints

1. The setup still resembles the type of plant I started with.
   This is not a problem if you modify the scope.
2. The PLC simulator I use only supports Modbus over TCP/Internet Protocol (IP) or RS232 and DF-1. To test other protocols, I will need another PLC simulator.
   Do you have other candidates?
3. Just one Supervisory and Control software package is used. If I would want to use other software, I would need to obtain another exploit. This process will take quite some time while, in my opinion, it doesn't provide much added value.
   You should first do the tasks with most impact and the least effort.

## Questions

1. Should any of the test-cases be removed?
   No.
2. Would you modify any of the test-cases?
   - No.
   - For the environmental changes "Use Modbus over RS232" and "Change the protocol to DF-1" you will need to interface an RS232 device on the other end. This is not TCP/IP traffic and you will need extra effort to simulate this. If you have time, it is ok to do this.
3. Do you have additional test-cases?
   - Have two channels open with the PLCs
   - Isolate the HMI from the internet / external access
   - Report to two HMIs
   - Include safety parameters
   - Make the liquid flow slower or faster
   - Enable/Disable autorun when Universal Serial Bus (USB) is connected (if you try spreading your malware through USB)
   - If possible create different types of processes. The idea is to see different types of values sent through Modbus/TCP.
     - Process that requires liquid level control and draining if the level is too high.
     - Process that requires heating and cooling of a liquid.
     - Process that requires measuring pressure.
     - Process that requires engaging motors for mixing of fluids in tanks.

- Add simulated traffic from different types of protocols.
- Try to replay previously captured traffic.
- Create an instable / very busy network with a lot of noise.
- Use a real tool that communicates over the network. It can interfere with the exploit/malware.
- Put the system in another VLAN.
- Use other IP address series (10.x instead of 192.168.x).

4. Is the current list constrained in any other way?

   - No.
   - The constraints you pointed out are not an issue for a targeted attack but they are for general malware. More about this in the next question.

5. Do you have any other remarks I should take into account?

   - No.
   - I think that if we take a targeted attack in mind, the current limitations will cover the possibilities. You will not perform such an attack without gathering sufficient information about the target. In that case the limitations will cover most variations. However if we take malware that targets different kinds of ICSs in mind, you would have to take; software types, versions, different OSs and other possibilities into account.

Appendix C

# Environmental changes

The questionnaire found in Appendix A was send to ICS and malware specialists who reviewed and completed the environmental changes. Their feedback can be found in Appendix B. The final list of environmental changes and their results will be listed here. Every environmental change is categorized and numbered to make it easier to refer to. The label of every environmental change consists of the category abbreviated to three characters (i.e., HMI, NET, PLC, PHY) and a number. An example would be 'PLC-3'.

After each environmental change the environment will be reverted back to the initial (clean) environment. This list contains a few environmental changes that are a combination of other environmental changes. It is not feasible to test all possible combinations of environmental changes.

## Human Machine Interface changes

We tested the effects of several changes in the OS as well as changes in the HMI settings and software changes.

HMI-1. Enable the firewall of the machine running the HMI

*Reason for testing*
When our malware wants to spread to other systems, it will first need to find them. A simple ICMP (ping) scanner is used to scan for potential targets in the network. If a firewalls blocks ICMP request by default then the malware will not be able to detect the machine and will not try to infect it.

*Hypothesis*
The firewall will block the ICMP request, the malware will not detect the machine and not try to infect it. The firewall will not affect the shortcut icon vulnerability used by the malware

*Test 1 - process to change the environment*

- Enable the firewall

*Test 1 - results*

- The firewall blocks ping request by default. This made the machine not visible during the ping scan and caused a vulnerable target to not be detected or infected. This also means that it would not be able to impact the system using the Interactive Graphical SCADA System (IGSS) remote exploits. The shortcut icon vulnerability still worked as usual.
- If the malware would be able to infect the machine, it would cause the usual impact.

Another scanning method is needed because it is impossible to change the firewall settings on other machines before we have infected it. Address Resolution Protocol (ARP) is used to get the Media Access Control (MAC) address associated to an already known IP address. ARP requests can also be used to scan the network. If a MAC address is associated to an IP address then a device is found.

*Test 2 - process to change the malware*

- Implement an ARP scanner
- Use the ARP scanner

*Test 2 - results*
The ARP scan is slower than the IP scan method. A scan over the IP address range of x.x.x.1 - x.x.x.256 took the ping-scanner 2 minutes and 6 seconds to complete which is almost 500ms per IP address if no

response is given and 1-2ms if a host responds. The ARP-scanner took 13 minutes and 9 seconds over the same range before it was finished. This means it takes three seconds for the ARP-request to time out if no response received. The ARP-scanner is 6 times slower compared to the ping-scanner. The scanners can be speeded by using multiple threads.

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
During the first test we have found that ping is blocked by the firewall by default. If we want to use this scanning method we need to know if firewalls are in place and which rules apply. We have implemented and tested another scanning method, based on ARP requests, that is not affected by the default firewall configuration. It would be unlikely that ARP requests are blocked. This would either hinder IP traffic from functioning or require static ARP entries. This makes the ARP scan more reliable than the ping scan.

HMI-2. Change the Operating System (OS) version on which the HMI runs
*Reason for testing*
IGSS runs in our environment on Windows XP SP3. This is an old operating system. It might be interesting to see how the malware reacts on a newer operating system. Note that IGSS does not run on Linux or MacOS based OSs, so our tests will only cover windows based systems.

*Hypothesis*
This change should not influence the exploits used according to their specifications.

*Test 1 - process to change the environment*

- Set up a new Virtual Machine (VM) with Windows 7 Service Pack 0 (SP0)
- Install IGSS

*Test 1 - results*

- The firewall in Windows 7SP0 blocks ping request by default. The firewall did not influence the malware when the ARP-scan was chosen as scan method. The shortcut icon vulnerability still worked.
- This change did not influence the impact

*Test 2 - process to change the environment*

- Set up a new VM with Windows 7 Service Pack 1 (SP1)
- Install IGSS

*Test 2 - results*
We encountered the same results as in test 1 with the exception that the shortcut icon vulnerability was patched and therefore not useable.

*System knowledge needed*
In the tests where we ran IGSS on newer OSs, i.e. Windows 7 SP0 and Windows 7 SP1, we noticed that the used OS affected the infection stage of the malware.

HMI-3. Change the IP address of the machine
*Reason for testing*
IP addresses can be dynamically assigned. The infection process of the malware would likely fail if the IP address was hardcoded. That is why the malware uses a scanner. The scanner scans a range of IP addresses for every network interface. The infected machine has two network interfaces, one for connections to the Internet and one for the local subnet. The default scan range is from x.x.x.1 to x.x.x.20 and this can be adjusted as needed.

*Hypothesis*
As long as the target's IP address is in the scanner's scan range, it would pose no problems to infection or impact. The default IP address of the machine is 192.168.1.14.

*Test 1 - process to change the environment*

- Change the IP address to 192.168.1.20
- Change the Local IP address in the used Modbus/TCP driver in System Configuration of IGSS.

*Test 1 - results*

- This change did not influence the infection
- This change did not influence the impact

*Test 2 - process to change the environment*

- Change the IP address to 192.168.1.150
- Change the local IP address in the used Modbus/TCP driver in System Configuration of IGSS.

*Test 2 - results*

- This IP address was not in the scanners range. No check was performed on the machine if it was vulnerable and it attempt was made to infect it. It was still possible to infect the machine with an infected USB drive.
- If the malware would be able to infect the machine, it would cause the usual impact

*Test 3 - process to change the malware*

- Set the scan range from x.x.x.1 to x.x.x.256.

*Test 3 - results*
The scan took 2 minutes and 6 seconds to complete. That is almost 500ms per IP address if no response is given and 1-2ms if a host responds.

- This change did not influence the infection.
- This change did not influence the impact.

*Test 4 - process to change the environment*
For this environmental change we want to change the IP address of the HMI machine to an IP address in the 10.x range. It is not possible to only change the HMI machine's address because it would leave it unable to communicate with the PLC or any other machine. Therefore we also change the PLC and the infected machine's IP addresses.

- Change the PLC machine address to 10.11.12.13 .
- Change the HMI machine address to 10.11.12.14 .
- Change the infected machine address to 10.11.12.15 .
- Change the IP address IGSS uses to communicate with the PLC.

*Test 4 - results*

- This change did not influence the infection.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
During this environmental change we learned that if an IP address is not in the range of the scanner, it would also not be scanned. If it is in the range it would be scanned. The scan range can be determined at runtime by requesting the subnet-mask of the local machine and OR-ing it to obtain the network prefix. All underlying IP addresses should be scanned

HMI-4. Enable Data Execution Prevention (DEP)
*Reason for testing*
Certain parts of a program contain just data and no code. DEP is used to mark these parts of memory as non-executable. This helps to prevent execution of data. DEP was introduced in Windows XP SP2 and DEP is only enabled for essential Windows programs and services by default.

*Hypothesis*
This should not affect the malware since our infection exploits are based on directory traversal vulnerabilities and do not contain shellcode.

*Process to change the environment*

- Turn on DEP for all programs and services

*Results*

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
In our case, the malware is not influenced by Data Execution Prevention (DEP).

HMI-5. Enable Address Space Layout Randomization (ASLR)
*Reason for testing*
Address Space Layout Randomization (ASLR) is one of the security features provided by the OS. It randomizes address offsets of (e.g. stack, heap, libraries) in the process's address space. This will make it harder to predict addresses that point to shellcode in the program.

*Hypothesis*
This should not affect the malware since our infection exploits are based on directory traversal vulnerabilities.

*Background information*
Since ASLR was not available in Windows XP and introduced in Windows Vista, it is not possible to use the default environment. The already set-up Windows 7 environment is used in this test. ASLR is on by default. However, this does not mean this test was cleared when Windows 7 was used as OS in environmental change HMI-2. Mark Russinovich's Process Explorer shows that while IGSSmaster.exe has ASLR enabled, dc.exe and IGSSdataServer.exe are not ASLR-aware build, i.e., they are not build with the /DYNAMICBASE linker flag.

*Test 1 - process to change the environment*

- Use the Windows 7 machine with default ASLR settings

*Test 1 - results*

- This change did not influence the infection using the IGSS exploits.
- This change did not influence the impact.

*Test 2 - process to change the environment*

- Use the previously setup Windows 7 environment.
- Install Windows update [1] which enables forced ASLR settings.
- Add forced ASLR entries for dc.exe and IGSSdataServer.exe to the register.

*Test 2 - results*

- IGSS could not start the dc.exe and IGSSdataServer.exe services. No supervision was possible.

As stated by Microsoft [1]; forcing ASLR for a program with the registry-key MitigationOptions set to 0x300 will prevent the program from loading if no reallocations are present.

*Lessons learned/System knowledge needed*
In this case ASLR could not influence the exploits because the services did not support ASLR. If an exploit is affected by ASLR, a type of shellcode named an 'egg-hunter' can be used to work around ASLR.

[1] `http://support.microsoft.com/kb/2639308/en-us`

HMI-6. Enable/Disable autoplay when a USB storage device is connected

*Reason for testing*
Autplay is a Windows feature that scans newly discovered media and performs an action / launches a program based on the contents. This environmental change will test the effect of autoplay on payload PAY-1 (infection by USB drive).

*Hypothesis*
When autoplay is set to 'Open folder to view files using Windows Explorer' then the malware is able to run without user interaction when the USB drive is inserted. The user has to open the infected folder if AutoPlay is set to another value.

*Process to change the environment* To set the autoplay functionality:

- For Windows 7:
  - Go to the Control Panel -> Hardware and Sound -> Autoplay .
  - Set the Mixed content value to 'Open folder to view files using Windows Explorer'.
- For Windows XP:
  - When the USB stick is inserted a 'Removable Disk' window is opened, check 'Always do the Selected action' and click 'Open folder to view files using Windows Explorer'.

*Results*

- The malware was able to spread using the shortcut vulnerability without user-interaction if the USB folder was automatically opened. Otherwise the user had to navigate to the infected folder manually. The other infection method was unaffected.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
The autoplay functionality is not configured by default. If autoplay is set to 'Open folder', explorer will automatically show the contents of the USB drive and the vulnerability would be executed. Otherwise it is up to the user to open the root folder of the infected drive. This is a plausible scenario since the user inserted the USB drive for a reason. It is also possible to automatically start malicious software in other ways using other techniques. A U3 USB stick can be used or a USB stick with a custom USB driver that simulates a keyboard can be used.

HMI-7. Change project paths used by HMI

*Reason for testing*
Payload PAY-2 used to infect the IGSS HMI contains hardcoded paths that define where it copies itself to and where it executes itself from. It copies itself to 'C:\Documents and Settings\All Users\Application Data\7T'. If these are changed it might influence the infection process

*Hypothesis*
The IGSS copy exploit will place the malware in another folder. The IGSS execute exploit will not be able to find the executable.

*Test 1 - process to change the environment*
The default project path is 'C:\Documents and Settings\All Users\Application Data\7T\IGSS32\V9.0'.

- Copy the project from the default project path to the desktop
- Make the default path inaccessible. We can do this by renaming the folder '7T' in the default project path to 'C:\Documents and Settings\All Users\Application Data\7T_renamed\IGSS32\V9.0'.

*Test 1 - results*

- The IGSS copy exploit created the folder '7T' and placed the malware there. Infection proceeded like normal and succeeded.
- This change did not influence the impact

*Test 2 - process to change the environment*

- Change to Windows 7SP1 because Windows 7 does not have the 'C:\Documents and Settings' path.

*Test 2 - results*

- The IGSS copy exploit placed the malware in the hidden folder 'C:\ProgramData\7T'. Infection proceeded like normal and succeeded.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
The exploits used did not require prior knowledge on system paths.

HMI-8. Increase/decrease HMI software update intervals
*Reason for testing*
The HMI software's update intervals indicate the time between each request to the PLC. A change in the time interval could possibly influence the MITM payloads. The default scan interval is 1000ms.

*Hypothesis*
This will have no effect on infection or impact.

*Process to change the environment*

- Scan intervals can be modified in IGSS during 'Design Mode' in 'System Configuration'. After each change the HMI interface will have to be updated. This is done by opening the Definitions module and saving the project.
  - Change scan interval to 2,000ms
  - Change scan interval to 500ms
  - Change scan interval to 10,000ms

*Results*

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
The frequency of which the HMI requests updates did not influence the infection or impact stage.

HMI-9. Run the HMI software as unprivileged user
*Reason for testing*
Functionality which requires administrator privileges do not run under an unprivileged account. If the malware uses such functionality it might fail during the infection or impact stage.

*Hypothesis*
Our malware is still able to perform the tasks as usual.

*Test 1 - process to change the environment*

- Create a new account with limited privileges.
- Copy the IGSS HMI project somewhere else because we can not alter the project used by the Administrator.
- Open the project in IGSS.

*Test 1 - results*

- This change did not influence the infection
- The malware ran a few seconds after the infection and then successfully exited. This was because it could not find the registry key used to check if IGSS was installed. We use an IGSS register that states that the current user has installed IGSS. The new account did not have the registry entry has all IGSS registers except the installed register because another user installed IGSS. We should have used another register for this check.

*Test 2 - process to change the malware*
Change the malware to use another registry key to determine if IGSS is installed.

*Test 2 - results*

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
In this case only unprivileged functionality was used by the malware. A privilege escalation exploit can be used if administrator rights are needed.

HMI-10. Add a secondary hot backup server
*Reason for testing*
Our malware now infects one HMI. Only one malware instance attempts to impact the environment at any given time. If the control system consisted of two HMIs, two HMIs would be infected. Would the malware interfere with the other infection?

*Hypothesis*
Both machines will be infected. The created impact will depend on the payload. The payloads without a MITM will perform as usual. Payload PAY-7 that performs a MITM will not work.

*Test 1 - Process to change the environment*

- Change the Station type to Station type to Server 1 in a Dualized System
- Create a new station like current station with the same drivers, interfaces and nodes but set it to Server 2 in a Dualized System. Also set the IP address to the IP address that will be used for the secondary machine.
- Clone the current HMI Virtual Machine
- Change the IP address on the secondary machine
- Change the MAC address on the secondary machine
- Start Sync Server Configuration on the secondary machine
- Start IGSS on the secondary machine and select Server 2 in a Dualized System as Station type

*Test 1 - Results*

- This change did not influence the infection. The malware infected both machines.
- This change did not influence the impact. The PLC address was ARP spoofed twice so effectively all traffic was send to the machine that was the latest infected. The malware on each machine responded only to their own HMI and filtered out the traffic of the other HMI.

*Test 2 - Process to change the environment*
This test determines how the malware reacts if it is unable to infect one of the two HMIs.

- Modify the malware so that it does not infect the second HMI by hardcoding its address in it.

*Test 2 - Results*

- This malware only infected one HMI.
- This change did not influence the impact. When PLC address was ARP-spoofed all traffic from both HMIs was send to the infected HMI. The malware on the infected HMI only responded to its own HMI and filtered out the traffic of the other HMI. The first HMI did not notice anything different but the other HMI did not receive any responses from the PLC since it send the request to the infected HMI and the malware did not respond. The second HMI gave a communication problem warning but it could not reconnect.

*Lessons learned/System knowledge needed*
The amount of HMI only affected the MITM payloads, so the lessons we've learned are mainly about them. If the malware can infect all HMIs, then the impact phase of a MITM attack should start synchronously across multiple machines that supervise a specific part. If the malware can not infect all HMIs but can communicate with them and spoof their ARP cache, it can still perform a MITM attack. It only needs to respond to their requests. This would mean that it should change its filter and optionally enumerate the affected HMIs at runtime. Otherwise; the HMIs the malware can not infect and communicate with, will be able to read the true state of the PLC or possibly be used to intervene.

HMI-11. Two single user supervisory and control machines (Report to two HMIs)
*Reason for testing*
This test looks a lot like environmental change HMI-10, but the machines are running totally separate. It is also possible to let the second single user machine only supervise part of the plant.

*Hypothesis*
This environmental change will lead to the same results as in environmental change HMI-10.

*Process to change the environment*

- Change the Station type to Single user
- Clone the current HMI Virtual Machine
- Change the IP address on the secondary machine
- Change the MAC address on the secondary machine
- Start IGSS on the secondary machine

*Results*

- This change did not influence the infection
- This change did influence the impact. The second machine was infected a second later than the first. The malware on the second machine started spoofing responses after the malware on the first machine performed the first action, i.e., opening the in-valve. This means the HMI on the second machine showed this action. As with the last environmental change the PLC address was ARP-spoofed twice so effectively all traffic was send to the machine that was the latest infected. The malware on each machine responded to their HMI.

*Lessons learned/System knowledge needed*
Same as in environmental change HMI-10.

HMI-12. When multiple PLCs are used, make each HMI supervise only part of the PLCs
*Reason for testing*
This environmental change tests how the malware responds to the scenario where it can only influence an unknown part of the setup. This environmental change also contains changes from PLC-1 'Used register addresses', HMI-11 'Two supervisory and control machines' and PLC-7 'use multiple PLCs'.

*Hypothesis*
The malware will infect both HMIs and infect and impact the system as usual.

*Process to change the environment*

- Use the environment created during the 'two single user supervisory and control system' HMI-11 environmental changes.
- Delete from HMI 1 the left half of the system (i.e., drum 2, pump 2, drum 3, pump 3 and out valve) since it will only supervise the right half. See Figure C.1.
- Delete from HMI 2 the right half of the system (i.e., drum 1, pump 1, drum 4, pump 4 and in valve) because it will only supervise the left half.
- Configure each of the drum and pump to a PLC like in environmental change PLC-7 'Change number of PLCs'.

*Results*

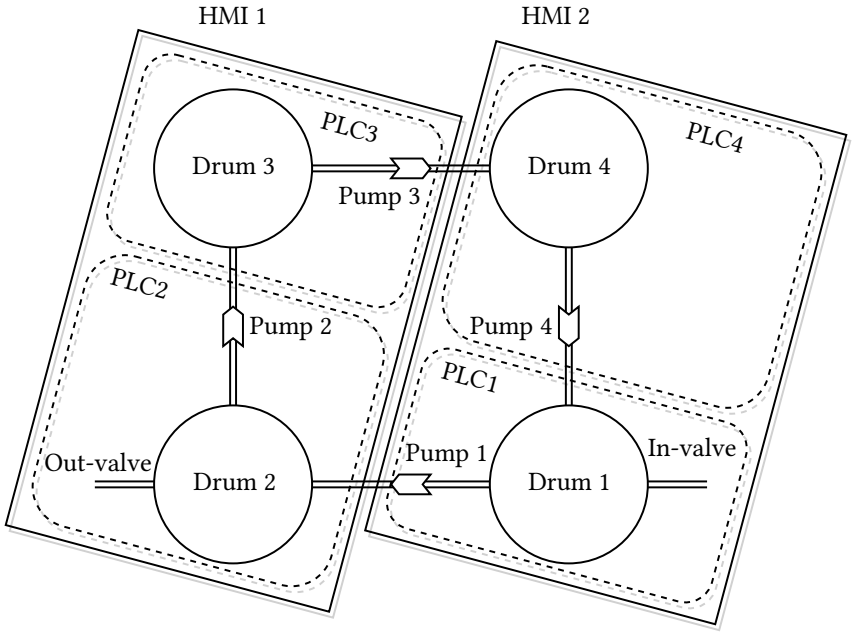



Figure C.1: Chemical plant where two HMIs supervise two PLCs each

- This change did not influence the infection
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Lessons learned/System knowledge needed*
The malware behaved the same as in environmental changes PLC-7 and HMI-11 combined and had the same result as mentioned in those environmental changes PLC-7 and HMI-11.

HMI-13. Have two channels open with the PLCs
*Reason for testing*
In this environmental change two connections will be opened between the HMI and the PLC; one normal connection and one backup connection. The backup connection will be used in case the normal connection does not respond.

*Hypothesis*
If both connections use the same network, the malware will perform as usual. But if both connections do not use the same network, the malware will not be able to disconnect the second connection between the HMI and PLC, giving the HMI a way to detect the MITM attack. All other payloads will perform as usual.

*Process to change the environment / Results*
The current HMI software does not support backup connections to the PLC. It is therefor not possible to test this environmental change.

HMI-14. Uninstall winpcap from the HMI
*Reason for testing*
Wireshark and winpcap were installed for debugging purposes. As discussed in environmental change 3.4.3, C does not support raw TCP sockets. Winpcap is used as an alternative to spoof packets.

*Hypothesis*
If the malware does not have a way to spoof packets, it will not be able to perform MITM attacks. All other functionality will still work.

*Test 1 - process to change the environment*

- uninstall winpcap

*Test 1 - results*

- This change did not influence the infection
- The malware did not know winpcap was not installed and could not be used to spoof packets. It decided to use the default payloads; PAY-7 with PAY-3 to overflow the drums while performing a MITM attack. When the malware requested all network interfaces from winpcap, the winpcap call returned an error and the malware decided to quit. MITM payloads PAY-6 and PAY-7 stopped working. The malware did not impact the environment.

*Test 2 - process to change the malware*

- implement a check to see if winpcap is installed
- if winpcap is not installed, use payload PAY-3 without MITM attack.

*Test 2 - results*

- This change did not influence the infection.
- Payloads PAY-3, PAY-4, PAY-5 and PAY-8 worked as usual.

*Lessons learned/System knowledge needed*
If the malware is not able to create raw sockets or use similar functionality, it is not possible to spoof traffic. Other payloads were not affected.

HMI-15. Isolate the HMI from the Internet or external access
*Reason for testing*
If the HMI is isolated or air-gapped from the outside then it would become hard, or nearly impossible to infect it. Isolating the HMI would mean the operational technology of the plant would be air-gapped with the outside world.

*Hypothesis*
The malware is not able to exploit the HMI and thus not impact the control system.

*Process to change the environment*

- Use the VirtualBox Network Manager to isolate the machine running the HMI from any other network except the control system network. Only the HMI and the PLC can be connected to this control system network. This way the plant is not connected to the outside world.

*Results*

- The malware was not able to detect any other machines and thus was not able infect the HMI. If a USB drive is available to the infected machine when the malware was running, it will be infected. It can infect the plant only if this infected USB drive was transported past the air-gap, and inserted in a machine at the plant. This does not have to be the machine running the HMI, but can also be a machine connected to it.
- Since the malware did not infect the plant, it could not impact it.

*Lessons learned/System knowledge needed*
If the plant is air-gapped, it is hard to infect it. The malware would have to physically travel past the air-gap on a storage device. This would require physical access to the plant and to a machine inside the plant. It would likely be impossible to create a backdoor to send information to a Command and Control server.

## Network changes

For these environmental changes, we changed the network properties and send noise and other traffic over the network to determine if it affected the malware

NET-1. Add simulated traffic

*Reason for testing*
This test will determine if the malware is affected by latencies or delays.

*Hypothesis*
This change will not affect the capability of the malware to infect or impact.

*Process to change the environment*
A script sends random traffic from the infected machine to the HMI and from the HMI to the PLC.

- Download netcat for windows. Netcat allows us to read from and write to network connections using TCP or UDP.
- Download Zero and Random device driver [2] which emulate '/dev/zero' and '/dev/random' in *nix like environments. This will be used to discard data and generate random data from the command line.
- The following command;
  `nc.exe -Lvvup 1234 > nul`
  can be used to receive traffic from port 1234 and throw it away. And
  `type \\.\random\ | nc.exe -vvu targetip 1234`
  can be used to send random traffic to a target on port 1234.

  One connection can generate around 3Mbps of traffic. To create multiple connections we can use a for loop. The following for loops are supposed to be one line but since it does not fit on one line, two lines are used.

  ```
  for /L \%i in (1234,1,1240) do
   @start "Receive on %i" "cmd.exe" "/c nc.exe -Lvvup %i > nul"
  ```

  can be used to listen on ports 1234 to 1240. And the command

  ```
  for /L %i in (1234,1,1240) do
   @start "Send to %i" "cmd.exe" "/c type \\.\random\ | nc -vvu targetip %i"
  ```

  can be used to send random traffic to a target on ports 1234 to 1240.
- Use these commands to send random traffic from the infected machine to the HMI.

*Results*
For this test we generated around 55Mbps of traffic with loop parameters `(1234,1,1250)`. On a sidenote; both sender and receiver had a high CPU load.

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
Other random traffic does not have influence on the malware.

NET-2. Instable network connection

*Reason for testing*
Since it is not possible to introduce noise on the Virtualbox virtual network, we'll try to simulate a very busy Local Area Network (LAN) with 100% network utilization. We will use the same method as in the previous environmental change, NET-1.

[2] `http://www.ltr-data.se/opencode.html/#ZeroDrv`

*Hypothesis*
The malware spreads 'slower' and it will take longer during the infection phase. This way of network simulation will also create a 100% CPU load on both sender and receiver.

*Process to change the environment*

- Assign three CPU cores to the HMI machine
- Assign two CPU cores to the infected machine
- Perform the same steps described in environmental change NET-1 with loop parameters `(1234,1,1290)`

*Results*
A 99% network utilization was created on the 100Mbps connection between the infected machine and the HMI machine. The generation of random numbers took quite some CPU time on the sender's machine. This resulted in a 100% CPU load on the sender's machine. The receiving machine also had a 100% CPU load because a lot of processed were created and this resulted in a lot of interrupts.

- This change did influence the infection speed. Every step of the process took at least two or three times longer but the functionality remained the same. The malware did infect the HMI.
- This change did influence the speed and latency of which it started the payload.

*Lessons learned/System knowledge needed*
A 99% network utilization and a 100% CPU load did slow the malware down but did not limit its functionality.

NET-3. Introduce traffic from a legitimate source
*Reason for testing*
In this environmental change network traffic from a legitimate source was added. The only legitimate ICS type of network traffic we're able to create at this moment is the traffic between two HMI servers.

*Hypothesis*
This will not reduce the impact of the malware.

*Process to change the environment*
In environmental change HMI-10 a secondary hot backup server was added. A primary and a secondary server are able to synchronized project data on port 12397. This port is also used by the malware.

- Add a secondary hot backup server.
- Start SyncSrvCfg.exe to synchronize between both servers. This service will communicate with the primary HMI machine on port 12397.

*Results*

- This change did not influence the infection.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
Introducing legitimate traffic did not reduce the impact of the malware.

NET-4. Replay some previously captured traffic
*Reason for testing*
In this environmental change we test if other traffic, including Modbus traffic interferes with the malware.

*Hypothesis*
It will not affect the malware.

*Process to change the environment*
In this test network traffic will be replayed while our malware tries to infect and impact the system. This requires a recorded network traffic file. The MITM payloads where able to spoof traffic for over an hour. During this tests we recorded the traffic with Wireshark in a .pcapng file. We use the recorded traffic file from the RESULT-7. test.

- Set up a Linux based virtual machine, for example Kali.
- Connect it to the virtual network.
- Use tcpreplay: `tcpreplay –intf1=eth0 -x=2.0 recordedtraffic.pcapng`

*Results*

- This change did not influence the infection.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
The malware filters traffic on IP addresses and ports. It was important here to set the filters correctly because the malware could respond to the replayed traffic otherwise. The ports used in the recorded traffic file where not the same because IGSS uses a random port every time it starts. This way the malware ignored all replayed traffic.

NET-5. Put the control system in another VLAN
*Reason for testing*
This environmental change looks a lot like environmental change HMI-15. The difference is that environmental change HMI-15 separates the infected machine from the plant physically while this environmental change only separates the infected machine from the plant logically.

*Hypothesis*
The same outcome as in environmental change HMI-15 since it does not matter how the plant is separated from the infected machine.

*Process to change the environment*

- Use the VirtualBox Network Manager to isolate the HMI from the infected machine by placing the HMI and the PLC machine on another network that is not connected to the network containing the infected machine. This way the HMI and the PLC are not connected to the infected machine.

*Results* Same as in environmental change HMI-15.

*Lessons learned/System knowledge needed*
The lessons learned are the same as in environmental change HMI-15 since it did not matter to the malware how the plant was separated from the outside world (physically or logically).

## PLC configuration changes

The environmental changes are based on adjusting settings of the PLC.

PLC-1. Change the register addresses used by the PLC
*Reason for testing*
The register addresses of the sensors and actuators are hardcoded in the malware. The malware knows which addresses are associated with the actuators. If we change the addresses used by the sensors and actuators, then the malware will still use the old addresses. If the malware writes to the wrong addresses (i.e., the addresses not used by actuators) then the physical process(es) will not be affected. This would, for example, mean that the malware would be unable to overflow or empty the drums.

*Hypothesis*
The malware will be able to spread to the HMI machine but the impact will be different if all register addresses are changed, then the malware will manipulate the wrong register addresses. This implies that the malware is unable to manipulate actuators and thus target the integrity of the processes. It will still be possible to attack the availability (payload PAY-8 will still succeed).

*Test 1 - process to change the environment*

- Change the PLC simulation script to use the same addresses, but with an offset of 1000.

- Change the HMI project definition to use the new addresses.

*Test 1 - results*

- This change did not influence the infection.
- The malware was able to compromise the supervision capabilities of the HMI using the MITM payloads but was unable to manipulate the actuators. The malware could still target availability by terminating the HMI process repeatedly with payload PAY-8.

*Test 2 - process to change the malware*
We will test if the malware can adapt to register changes by maturing it and without introducing new knowledge. We developed functionality that analyzes the environment for a specified time. With this information we can create an impact on the integrity of the environment. The malware will analyze the environment until it has analyzed a predefined number of packets.

- When the HMI is infected, the first step would be to listen to every connected PLC. The malware allocates and initializes the variables that are used to analyze the traffic.
- The malware analyzes the traffic. It looks at the memory registers that have been written and the minimum and maximum values that are read. Finally, it records a response packet which will be used later to spoof responses.
- The malware writes to every writeable memory register either its minimum or maximum recorded value.

*Test 2 - results*

- This change did not influence the infection.
- The malware is able to write either the highest or lowest recorded value to each writeable PLC register. It is not possible to define the impact on the ICS in advance, because the state created by the malware is random.

*Lessons learned/System knowledge needed*
If we do not know what registers are used to read from and write to, we need to analyse the traffic or write to all addresses. During the tests, we analyzed the traffic between the PLC and HMI and determined which values were needed to be spoofed and which addresses were connected to actuators.

The impact phase relied on the assumption that the smallest or largest values are potentially the most dangerous values for actuators. With the knowledge gained from analyzing the packets during runtime we were able to create a random potentially dangerous state. Another similar possible implementation could be to write random, zeros or maximum values to the PLC. An increased probability to create impact can be achieved if we know that larger values are more/less likely to create impact than small values. More consistent results are created when the registers are set to the highest recorded value.

The malware analyzed the traffic when the PLC communicated to the HMI through Modbus/TCP. The malware might not be able to analyze the traffic, if the traffic is encrypted or when another protocol or interface is used. We can not determine the possibilities for the other cases yet. We determine if it is also possible to analyse serial connection traffic when we connect the PLC to the HMI with a serial connection in environmental change PLC-5.

PLC-2. Change the IP address
*Reason for testing*
If the IP address of the PLC was hardcoded it could prevent the payload from executing. When the malware wants to determine the IP address of the PLC it will listen to all TCP/IP traffic between the current machine and port 502 of another machine. It uses a filter to block out all other network traffic. It saves the connection information like the port, IP address and MAC of the source and destination in a ‘connection’ struct.

*Hypothesis*
Both infection and impact will execute as usual.

*Test 1 - process to change the environment*

- Change the IP address the PLC listens to 192.168.1.196 .
- Change the IP address IGSS uses to communicate with the PLC.

*Test 1 - results*

- This change did not influence the infection.
- This change did not influence the impact.

*Test 2 - process to change the environment and results*
The environmental change to test IP addresses to an address in the range of 10.x has been described in environmental change HMI-3 in test 3.

*Lessons learned/System knowledge needed*
The supervisory software interacts with the PLC. If we know the process that is responsible for communication with the PLC it is possible to get communication details like IP addresses and ports with the netstat command.

PLC-3. Change the port
*Reason for testing*
The default Modbus port is 502. When the malware tries to determine the IP address of the PLC it will listen to traffic destined for port 502. All other traffic is filtered out. This filter will likely influence the malware when a non-standard port is used.

*Hypothesis*
Infection will proceed a usual but it will not be able to find a PLC and thus not impact the environment.

*Test 1 - process to change the environment*

- Change the port the PLC listens to (port 507).
- Change the port IGSS uses to communicate with the PLC.

*Test 1 - results*

- This change did not influence the infection.
- No PLC was found and therefor the malware could not communicate with the PLC to impact the environment. Payload PAY-8 was not affected.

*Test 2 - process to change the malware*

- Change the filter used by the malware so that it does not filter traffic on port 502. The only filter remaining filters on TCP/IP traffic.

*Test 2 - results*

- This change did not influence the infection.
- This change did not influence the impact.

*Lessons learned/System knowledge needed*
It is convenient if port 502 is used for Modbus traffic, but this information can be gathered during runtime. In environmental change PLC-7 we discuss other possibilities to enumerate connected PLCs at runtime.

PLC-4. Change response latency
*Reason for testing*
The malware might be sensitive to late responses from the PLC. The standard latency of the PLC can be set to a value between 0 and 10000ms and is set to 100ms in our default setup.

*Hypothesis*
This change will not affect the capability of the malware to infect or impact.

*Process to change the environment*

- Change the PLC responsiveness to:

  - 0ms latency
  - 20ms latency
  - 50ms latency
  - 500ms latency
  - 3,000ms latency
  - 10,000ms latency

*Results*

- This change did not influence the infection
- This change did not influence the impact of any payload except for payload IV with the MITM type 1 attack that simply tries to respond faster than the PLC. Delays larger than ~20ms were no problem, but delays smaller proved to be too fast for the malware to respond. This delay might be due to the CPU-scheduler, scheduling the malware after the HMI software. As said before; all other payloads, including the other MITM, worked fine.

*Lessons learned/System knowledge needed*
We did not see any differences in the malware's behavior during the change of the response delay.

PLC-5. Use Modbus over RS232 instead of TCP/IP

*Reason for testing*
The malware has no build in support for the RS232 interface (serial port).

*Hypothesis*
The malware is not able to find any connections to PLCs. It will just exit without trying to cause an impact. If support is added, it will not be able to communicate to the PLC since the HMI software is already connected to the serial port.

*Test 1 - process to change the environment*

- Add a serial interface (COM1) between the HMI machine and the PLC machine.
- Set the PLC simulator to use Modbus over RS232 on port COM1 with a baud rate of 19200, Parity 'Even', 8 data bits, 1 stop bit and RTS control to 'Enable'.
- Open the System configuration module in the HMI software. Add a new 'Modicon Modbus protocol' driver. Configure this driver to connect to COM1 with the same settings as the PLC. Add a node and set the node number to 1.
- Open the Definitions module and set every sensor and actuator to use the new driver and node 1.

*Test 1 - results*

- This change did not influence the infection.
- The malware was not able to find any connected PLC and did not impact the system. Payload PAY-8 still worked.

*Test 2 - process to change the malware*
The malware should be able to connect and communicate with a PLC over RS232 using Modbus.

- Add functionality to list all available serial ports.
- Add functionality to determine the port that is used by the HMI software.
- Add functionality to use Modbus over serial port.

*Test 2 - results*

- This change did not influence the infection.
- The malware was not able to connect to the serial port when the HMI software was running.

*Test 3 - process to change the malware*
A connection should be made even when the HMI software runs.

- The malware should be able to terminate the processes that communicate over the serial port. The processes that connect to the PLC are 7tdrv.ex and dc.exe.

*Test 3 - results*

- This change did not influence the infection.
- The malware was able to connect to the serial port and communicate with the PLC to overflow the drums. Since the malware did not release the serial port connection, it left the HMI unable to reconnect. If the HMI software can not reconnect to the serial port after a few, probably one or two, tries it will stop trying and give a warning on the initial 'welcome'/'configuration' screen. It is likely that this screen is not visible but the HMI is shown instead because this screen can be used to supervise and control the processes. The HMI does not give any warning/alarm. This means that when the HMI screen is shown, the only way to see the connection loss would be to see a small change in the IGSS icon in the taskbar. The HMI will show the last remembered state during the attack until the supervision capabilities of the HMI are restarted. This is a Denial of Service (DoS) attack since it is not possible to communicate with the PLC or modify its registers.

*Test 4 - process to change the malware*
This test is to determine if there is a way to keep both HMI software and the malware connected to the same PLC without either of them giving warnings.

- Change the malware to write a harmful state to the PLC in a loop. Start this loop by terminating the supervisory processes that communicate with the PLC over the serial port, connect to the PLC over this serial port and write the harmful state. The final steps would be to close the connection and sleep for five seconds.

*Test 4 - results*

- This change did not influence the infection
- By closing the serial port every time after user we allow the HMI software to restart the communication services and reconnect to the PLC over the serial port. This way both software and malware can use the same serial connection. The HMI software will give no warnings or errors. The HMI is able to show the values from the PLC and modify registers although it is not very responsive. It is not very responsive because it only updates in a short time span and when the HMI is outdated it is not possible to modify actuators. The HMI can be used to manipulate the PLC registers to delay the impact of the malware but since this functionality is only available for a short period of time, it is not possible to completely protect from it.

*Lessons learned/System knowledge needed*
It is possible to connect to the PLC and keep the HMI working without errors or warnings. Modbus over RS232 supports two transition modes; RTU and ASCII. This affects the number of bits per byte and coding system. The baud rate is also required to communicate with the PLC but does not have to be known in advance. According to the Modbus specification [19] the baud rates 9600 bps and 19.2 Kbps are required to implement and 19.2 Kbps is the required default. The number of possible baud rates is also limited and it is possible to try and guess the correct baud rate in a limited number of tries. The node id can be determined by requesting information from the PLC. Knowledge about the environment is needed in advance since it is not possible to have a serial port opened twice without installing new drivers, changing settings or physically changing cables. We can analyze the state of the PLC over a period of time with the same mechanism as we use to write to the PLC. First disconnect the HMI from the serial port, connect to the serial port, read the PLC state and then close the connection to give the HMI a chance to connect. We can not sniff the write messages performed by the HMI because these can only happen when the HMI is connected.

PLC-6. Change protocol to DF-1
This protocol uses the serial port as the previous environmental change (PLC-5) where Modbus is used as protocol. The difference between the two environmental changes is the protocol. No implementation of the DF-1 in C is available for Windows based systems. It would be possible to develop the DF-1 protocol

for Windows but since it does not have more security features than Modbus/RS232 it will not create any different knowledge learned with regard to the previous environmental change.

*Lessons learned/System knowledge needed*
DF-1 is ASCII character oriented protocol with the following parameters; 8 data bits, no parity, and a maximum baud rate of 19200. Since most parameters are fixed, we only need to guess the baud rate.

PLC-7. Change number of PLCs

*Reason for testing*
The current version of the malware is programmed with support for only one PLC. The malware would not be able to manipulate the actuators of the other PLCs. The malware listens to the traffic and uses the first connection and PLC that it records. If the PLC does not contain any actuators, the malware would not be able to attack the integrity of the system.

*Hypothesis*
The malware is not programmed to detect more than one PLC. It will not detect the other PLCs and not be able to impact the ICS.

*Test 1 - process to change the environment*

- Clone the PLC VM three times. Now we have four PLCs.
- Configure the HMI to connect to four PLCs.
- Configure the HMI to request the data of Drum 1 and Pump 1 from PLC 1, data from Drum 2 and Pump 2 from PLC 2, etc. See Figure C.2.
- Since the VBscript running on PLC 1 can not simulate the behavior of all PLC, we need another method to simulate the physical environment. The physical environment can be simulated with a C program that behaves similar to the VBscript. It reads the used registers from all PLCs, calculates new values and writes the modified values back.

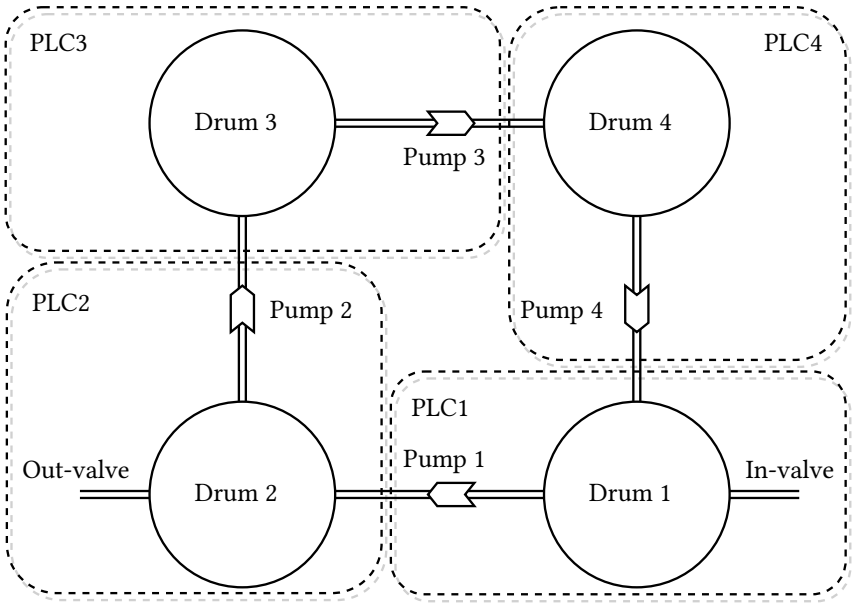



Figure C.2: Chemical plant with four PLCs

*Test 1 - results*

- This change did not influence the infection
- This change did influence the impact. The malware performed the same actions as usual. After the malware recorded one packet from PLC 1, it performed the ARP-spoof and then it manipulated several registers on for PLC 1 which caused the in-valve to open, and Pump 1 to go to automatic.

The states of the other pumps and valve remained unchanged. The result was that drum 1 and 2 were filled and eventually overflowed.

The result of the test was more luck than wisdom because the first PLC contained an important valve, e.g., the in-valve, the addresses of the in-valve and pump 1 where the same and the malware had knowledge about how the valves and pumps worked. The goal now would be to change the malware so it would not use this information any more.

*Test 2 - process to change the malware*
We will change the malware so that it identifies all PLCs, analyzes the traffic and finally interact with the actuators while it performs a MITM attack.

- The first step would be to identify the connections to the PLCs. This can be done in several ways, three of those are: we can search through the HMI software's project files to look for connections and properties. The second way is to sniff traffic to find out which connections are established to the default Modbus port. Another way is to list all connections established by the HMI's communication driver (with the netstat command). The first options needs to know which HMI system software is running and takes more time to define for multiple HMI software packages. The second option assumes that a specific protocol is used on the default port for all PLCs while the third option needs to know the name of the HMI's driver. The last option would only needs the name of the service that communicates with the PLCs and it is possible to predefine several options for several HMI software packages that can be tested at runtime with the same accuracy. The last option is chosen to implement.
- The next step would be to listen to every connection to allocate memory and initialize the variables and buffers we use to analyze the traffic.
- To analyze the traffic we record the memory registers that have been written to and the minimum and maximum values of the next $n$ samples. Finally we record a response packet for each request. These responses will be used to spoof responses after the ARP-spoof.
- Modify the ARP-spoof functionality so it sends ARP packages to all PLCs.
- Modify the response part so it uses the recorded packets to respond to each corresponding PLC.
- Add functionality to send to each PLC for every writable memory register either its minimum or maximum recorded value.

*Test 2 - results*

- The infection part worked as usual
- The malware behaved as it should; it identified all PLCs, analyzed the traffic, wrote to the PLCs and performed the ARP spoof. The result was that the C program, that simulated the physical environment, could not communicate to the other PLCs any more and was therefor unable to simulate the physical environment.

*Test 3 - process to change the environment*
We now need to modify the environment and physical environment simulation program so that it will keep functioning during the attack.

- create a new local network for the sole purpose of simulating the physical environment.
- connect it to the machines that simulate the PLCs and assign a unique IP address to every machine.
- modify the physical environment simulation program to use the new IP addresses. Now it will not be affected any more by the ARP spoof.

*Test 3 - results*

- This change did not influence the infection.
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Lessons learned/System knowledge needed*
During this test we needed a way to identify the connections between the HMI and PLCs. We can get a list of all connections if we know something about the IP addresses or ports in advance or if we know the process name that communicates with the PLCs.

The same lessons as in environmental change PLC-1 'PLC change register address' test are applicable here; in this environmental change we also analyzed the network traffic and created a random potentially dangerous state.

## Physical process changes

When we tested how changes in the physical environment affected the malware, we mostly changed the workings or the quantity of the sensors and actuators in the environment.

PHY-1. Change number of pumps
*Reason for testing*
The number of pumps is hardcoded in the malware. We test if the amount of pumps influence the malware.

*Hypothesis*
The malware will ignore the fifth pump and create the same impact as before. The analysis functionality developed during environmental change PLC-1 'Change memory addresses used' will enable the malware to create a random potentially dangerous system state.

*Test 1 - process to change the environment*

- Open the Definition module where the HMI can be changed.
- Add a new pump, name it 'Pump_5' and add pipes to visually connect the pump from Drum 1 to Drum 3.
- Update the physical environment simulation script to support the fifth pump.

*Test 1 - results*

- This change did not influence the infection
- This change did not influence the impact

*Test 2 - process to change the malware*

- Use the analysis functionality developed in environmental change PLC-1 'Change memory addresses used'.

*Test 2 - results*

- This change did not influence the infection
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Test 3 - process to change the environment*

- Remove Pump 2

*Test 3 - results*

- This change did not influence the infection
- Same impact as in the previous two tests

*Lessons learned/System knowledge needed*
The amount of pumps did not have influence on the malware. If the in-valve is open and the out-valve is closed, the total amount of chemicals in the system will increase and (at least) one drum will overflow eventually. The same holds for the case if the in-valve is closed and the out-valve is opened; the total amount of chemicals in the system decreases and at least one drum will eventually be empty.

PHY-2. Change number of drums
*Reason for testing*
The number of drums is hardcoded in the malware. We will test if a change in the number of drums can reduce or diminish the effect of the malware.

*Hypothesis*
The malware will not be influenced by the number of drums.

*Test 1 - process to change the environment* If we add a drum, it would make sense to also add a connection between the newly added drum and the other drums. To connect the new drum to the other drums, pipes and a pump are added.

- Open the Definition module where the HMI can be changed.
- Add a new drum, name it 'Drum_5' and add a pump and pipes to connect the drum to from Drum 4 and Drum 1. Now we have a circle of five drums instead of four.
- Update the physical environment simulation script to support the fifth drum and pump.

*Test 1 - results*

- This change did not influence the infection
- This change did not influence the impact

*Test 2 - process to change the environment*

- Start from the clean environment (where we have four drums).
- Remove Drum 4. Now we have three drums.

*Test 2 - results*

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
The malware does not need to know how many drums are present in the plant. If a drum is added, its state will also be spoofed during the MITM attack.

PHY-3. Change number of in- or out-valves
*Reason for testing*
The malware is able to overflow the system by putting the opening the in-valve and the closing the out-valve. The state of the other actuators is of minor importance. If the total amount of chemicals in the system increases, it will overflow eventually.

*Hypothesis*
The malware is able to impact the system in the same way as before. If an attack, intended to overflow all drums, is detected it can be prevented by opening the second out-valve. This will create a balance between the amount of chemicals flowing in and out. This way the malware will not be able to overflow the drums any more. A similar scenario holds when a second in-valve is present and the malware wants to empty all drums.

*Test 1 - process to change the environment*

- Open the Definition module where the HMI can be changed.
- Add a new valve and add pipes to visually connect the valve to Drum 3.
- Update the physical environment simulation script to support the second out-valve.

*Test 1 - results*

- This change did not influence the infection
- This change did not influence the impact if the attack was not detected. If it was detected and it was an overflow-type attack, it could be prevented by opening the second out-valve. However; the actuators, including the second out-valve, can not be manipulated from the infected HMI.

*Test 2 - process to change the malware*

- Use the analysis functionality developed in environmental change PLC-1 'Change memory addresses used'.

*Test 2 - results*

- This change did not influence the infection
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Test 3 - process to change the environment*

- Open the Definition module where the HMI can be changed.
- Add a new valve and add pipes to visually connect the valve to Drum 4.
- Update the physical environment simulation script to support the second in-valve.

*Test 3 - results*
In this test we use the unmodified malware that uses hardcoded values.

- This change did not influence the infection
- This change did not influence the impact if the attack was not detected. If it was detected and it was an empty-type attack, it could be prevented by opening the second in-valve. However; the actuators, including the second in-valve, can not be manipulated from the infected HMI.

*test 4 - process to change the malware*

- Use the analysis functionality developed in environmental change PLC-1 'Change memory addresses used'.

*Test 4 - results*

- This change did not influence the infection
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

If the in- or out-valve is removed from the initial setting it would make the system lose critical functionality. Therefor these tests are not performed.

*Lessons learned/System knowledge needed*
It can be crucial for a predefined attack to have knowledge of all important actuators. The malware can use a scanner to determine which actuators are available

PHY-4. Change amount of liquid drums can hold
*Reason for testing*
This test determines if the malware depends on maximum liquid levels.

*Hypothesis*
This will not influence the malware but it will influence the time it takes to fill and overflow the drums since the flow speed of the liquids is still the same.

*Process to change the environment*

- Open the Definition module where the HMI can be changed.
- Define for every drum the maximum liquid level to 250 and set the alarms and warnings accordingly.
- Update the physical environment simulation script so that drums can hold up to 250 liquid.

*Results*

- This change did not influence the infection
- This change did not influence the impact

*Lessons learned/System knowledge needed*
The amount of liquid a drum can hold is not important to know in advance. The amount of liquid a drum can contain only affects the time it takes to put the system in a critical state.

PHY-5. Change working of the pumps
*Reason for testing*
The values of the settings of the pumps are hardcoded in the initial version of the malware. The default settings of the pump are 0 for Auto, 1 for Off and 2 for On. If this is different in the physical environment then the malware can put a pump in a different state than intended.

*Hypothesis*
The malware will write the same values as before to the pumps.

*Test 1 - process to change the environment*

- Update the physical environment simulation script with the following settings:
  - Assign 0 to 'Off' (0 is by default 'Auto')
  - Assign 1 to 'On' (1 is by default 'Off')
  - Assign 2 to 'Auto' (2 is by default 'On')

*Test 1 - results*

- This change did not influence the infection
- The malware tries to overflow the drums and sets most pumps now to 'Off' instead of 'Auto' and Pump 4 to 'On' instead of 'Off'. This resulted in Drum 1 overflowing and Drum 4 being emptied.

*Test 2 - process to change the malware*

- Use payload PAY-4 'empty all drums'.

*Test 2 - results*

- This change did not influence the infection
- The malware tries to empty all drums and sets most pumps now to 'Auto' instead of 'On' and Pump 2 to 'On' instead of 'Off'. This resulted in Drum 2 being emptied at a fast rate while the other drums more or less kept their liquid levels.

*Test 3 - process to change the malware*

- Use the analysis functionality developed in environmental change PLC-1 'Change memory addresses used'.

*Test 3 - results*

- This change did not influence the infection
- The malware is able to write either the highest or lowest recorded value to each writable PLC register. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Lessons learned/System knowledge needed*
The valves are also used to create an impact and if the malware can use the pumps correctly it can optimize the impact. For instance, if we want to empty a drum we can open the out-valve but if we want to empty all drums we would have to enable the pumps. Although it is not necessary to know how the actuators work, it is very convenient to know. Without this knowledge we would have to analyze and guess the potentially harmful state of each actuator and thus decreasing the success-rate.

PHY-6. Make the liquid flow slower/faster
*Reason for testing*
This environmental change will test if the flow speed of the liquid affects the malware. The valves let 1 liquid unit flow in or out and pump pumps 2 liquid units by default.

*Hypothesis*
If the liquid flow is faster, the critical state is reached faster. If the liquid flow is slower, it will take a longer time to reach a critical state.

*Process to change the environment*

- Update the physical environment simulation to change the valve and pump flow speeds
- Set the valve flow speed to 3.
- Set the pump flow speed to 4.

*Results*

- This change did not influence the infection
- This change did influence the time it took before the payload emptied or overflowed the drums.

*Lessons learned/System knowledge needed*
No prior system knowledge is needed about the liquid flow speed. The liquid flow speed only affects the time it takes to impact the system.

PHY-7. Add a process that requires liquid level control and draining if the level is too high. (Include safety parameters)
*Reason for testing*
In this test a safety system is included to prevent drums from overflowing or being emptied. The safety system should prevent the physical environment from reaching a critical state.

*Hypothesis*
The malware will not be able to overflow or empty any drum.

*Process to change the environment*

- Include a safety system in the physical environment simulation script. This safety system should have priority other write commands and it should not be accessible from the HMI.
- The safety system will check the chemical level of every drum. If the chemical level of the current drum is higher than the maximum level safety threshold, set the pump to 'auto'. If all drums are almost full, close the in-valve. If the chemical level of the current drum is lower than the minimum level safety threshold, set the previous pump to 'on'. If all drums are almost empty, close the out-valve.
- The safety system uses 5 units as the minimum level safety threshold and 95 units as the maximum level safety threshold.

*Test 1 - results*

- This change did not influence the infection.
- The malware could not overflow a drum. The chemical levels did not exceed 1 unit more than the maximum chemical level safety threshold.

*Test 2 - process to change the malware*

- Use payload PAY-7 in combination with PAY-4.

*Test 2 - results*

- This change did not influence the infection.
- The malware could not empty a drum. The chemical levels did not drop lower than 1 unit below the minimum chemical level safety threshold.

*Lessons learned/System knowledge needed*
The safety system designed to prevent an overflow or empty attack successfully prevented these attacks.

PHY-8. Add a process that requires engaging motors for mixing of fluids in the tanks
*Reason for testing*
This environmental change adds different types of processes to the environment. The malware is not programmed to control these processes. This environmental change test if the malware is capable of impacting the environment with this extra process.

*Hypothesis*
The malware will still infect the HMI as usual and continue executing as usual. The impact depends on the case if the addresses for the valves (or pumps) changed.

*Test 1 - process to change the environment*
For this environmental change a process that mixes different chemicals is added. This requires different types of chemicals. We support two different unmixed chemicals (i.e., 'A' and 'B') and a mixed chemical 'C'. A second in-valve will be created as input for chemical 'B' while the already existing in-valve is used for chemical 'A'. The chemicals do not mix naturally and therefor have to be mixed by a mixer. A mixer will mix one unit of chemical 'A' with one unit of chemical 'B' to create two units of the mixed chemical. The out-valve can be used to let mixed chemicals flow to the next process. Figure C.3 shows this process. In this scenario the mixed chemicals are the heaviest and chemical 'A' is the lightest. The pumps pump the heaviest chemicals to the next drum; if no mixed chemicals 'C' are present in the drum it will pump chemical 'B' and if those are not present, chemical 'A'. The out-valve works the same way and will flow unmixed chemicals to the next process if no mixed chemicals are available.

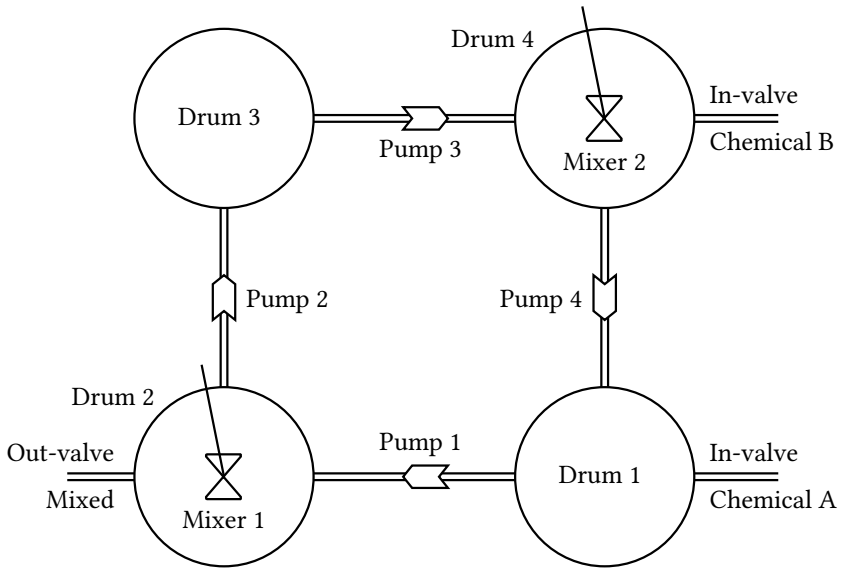



Figure C.3: Chemical plant where chemical 'A' is mixed with chemical 'B' to the mixed chemical C

To obtain the described environment we can implement the following changes:

- Open the Definition module and apply the following changes to the HMI:
  - add a second in-valve for drum 4 according to the steps described in environmental change PHY-3.

  - add for every drum three level-bars, one to view the amount of chemical A in the drum, one for chemical B and one for the mixed chemicals.
  - add for drums 2 and 4 a mixer which will mix chemicals A with chemicals B to obtain the mixed chemical.

  After applying these changes, our HMI will resemble the HMI in screenshot C.4 which was taken during test 1.

- Update the physical environment simulation script with the following settings:
  - Add three chemical level variables, i.e., A, B and C, to the program which hold information about the levels of chemicals A, B and the mixed chemical in the current drum.
  - The already existing chemical level variable will be used as the total chemical level of the drum. This will be the amount of chemical A plus the amount of chemical B plus the amount of the mixed chemical C.
  - Edit the already existing in-valve of drum 1 to add chemicals of type A.
  - Add a new in-valve which enters chemicals of type B intro drum 4.
  - Edit the out-valve to be the exit point of the process for chemicals of type C. If no chemicals of type C are present in Drum 2, unmixed chemicals will flow to the next process.
  - Add two mixers, one in drum 2 and one in drum 4. These mixers mix 1 unit of chemical A with 1 unit of chemical B to obtain 2 units of the mixed chemical if turned on.

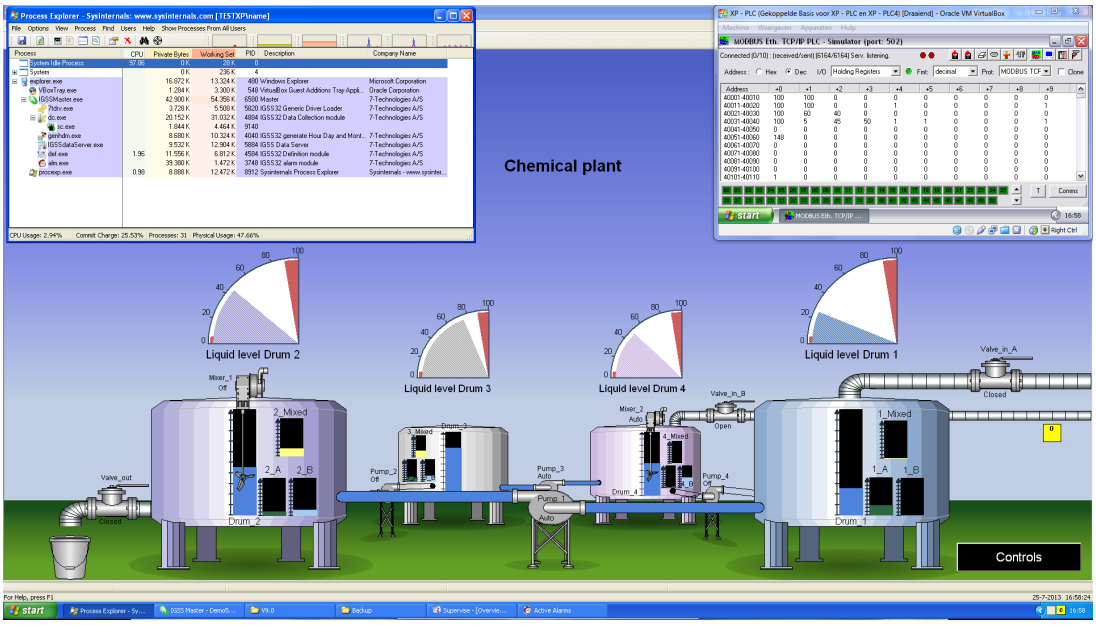



Figure C.4: Overview chemical plant HMI with mix processes for multiple chemicals

*Test 1 - results*

- This change did not influence the infection.
- The in-valve, out-valve and the pumps are still at the same address and work the same way as in earlier tests. The malware does not know that an additional process is added and will execute as if it is the normal plant. It opened the in-valve and chemicals A flowed into the system. The malware then modified the state of the pumps, filled all the drums with chemicals A and overflowed the drums.

Under normal circumstances chemicals will be mixed before they leave the process. Now that we have unmixed chemicals, it would be interesting to see if we can let unmixed chemicals flow to the next process.

*Test 2 - process to change the malware*

- Use payload PAY-7 in combination with PAY-4.

*Test 2 - results*

- This change did not influence the infection.
- The in-valve, out-valve and the pumps are still at the same address and work the same way as in earlier tests. The malware emptied all drums completely and unmixed chemicals left the process before they were mixed.

*Test 3 - process to change the malware*
The last results were obtained because the environment was largely the same and no additional modifications on the added process were required to create impact. This test determines the impact if the analyzer is used.

- Use the analysis functionality developed in environmental change PLC-1 'Change memory addresses used'.

*Test 3 - results*

- This change did not influence the infection.
- The malware was able to write either the highest or lowest recorded value to each writable PLC register. The malware has a higher change to overflow the drums since the environment contains two in-valves and just one out-valve. It is not possible to define the impact on the ICS in advance because the state created by the malware is random.

*Lessons learned/System knowledge needed*
It is possible to just target one process in a control system where multiple processes are controlled. For example; we can attack the system by emptying or overflowing the drums. We can also attack the system by letting unmixed chemicals flow to the next process or by creating a dangerous chemical ratio between two or three chemicals in a drum. We are able to create a potentially harmful state in test 3 for both processes.

Appendix D

# VBscript physical environment code

This appendix contains the physical environment emulation code programmed in Visual Basic script. The functions provided by the PLC simulator (i.e., 'getregistervalue' and 'SetRegisterValue') are underlined.

```
dim NROFDRUMS, MAX_LEVEL, PUMP_AUTO_TRES, PUMP_FLOW, VALVE_FLOW
dim VALVE_OPEN, VALVE_CLOSE, PUMP_AUTO, PUMP_OFF, PUMP_ON

NROFDRUMS       = 4         ' Number of drums controlled by the system
MAX_LEVEL       = 100       ' Max chemical level in a drum
PUMP_AUTO_TRES  = 80        ' Drum chemical level for pumps set to auto to activate
PUMP_FLOW       = 2         ' Speed of which the pump pumps liquids
VALVE_FLOW      = 1         ' The flow speed of an open valve

VALVE_OPEN      = 0
VALVE_CLOSE     = 1
PUMP_AUTO       = 0
PUMP_OFF        = 1
PUMP_ON         = 2

SimulatePhysicalEnvironment()

Sub SimulatePhysicalEnvironment()
    dim    level,    pump,    valve     ' Level state,    Pump state,    Valve state
    dim reglevel, regpump, regvalve     ' Level register, Pump register, Valve register

    dim nreglevel                       ' Level register of next drum
    dim drum, ndrum                     ' Current drum,     next drum

    dim init                            ' If all values need to be initialized

    init = getregistervalue(3,100)      ' Get if all values are initialized
    if init = 0 then                    ' If init is 0 (false)
        InitPhysicalEnvironment()       ' Initialize environment
        SetRegisterValue 3, 100, 1      ' Set init to 1 (true)
    end if

    for drum = 0 to (NROFDRUMS-1)
        reglevel = drum*10+0
        regpump  = drum*10+5
        regvalve = drum*10+9

        ndrum = (drum + 1) Mod NROFDRUMS
        nreglevel = ndrum*10+0

        pump = getregistervalue(3,regpump)              ' Get pump state
        if pump = PUMP_ON then                          ' Pump On
            Call PumpWater(reglevel, nreglevel, PUMP_FLOW)
        elseif pump = PUMP_AUTO then                    ' Pump Auto state
            level = getregistervalue(3,reglevel)
            if level > PUMP_AUTO_TRES then
                dim nlevel
                nlevel = getregistervalue(3,nreglevel)
                if level > nlevel then
```

```
50                      Call PumpWater(reglevel, nreglevel, PUMP_FLOW)
51                  end if
52              end if
53          end if
54
55          valve = getregistervalue(3,regvalve)
56          if drum = 0 and valve = VALVE_OPEN then
57              Call FillDrum(reglevel, VALVE_FLOW)
58          end if
59          if drum = 1 and valve = VALVE_OPEN then
60              Call LeakDrum(reglevel, VALVE_FLOW)
61          end if
62      next
63  End Sub
64
65  Function LeakDrum(reglevel, amount)
66      dim level
67      level = getregistervalue(3,reglevel)
68      if level < amount then
69          amount = level
70      end if
71      SetRegisterValue 3, reglevel, level-amount     ' Pump water out of current drum
72      Exit Function
73  End Function
74
75  Sub FillDrum(reglevel, amount)
76      dim level, overflowed
77      level = getregistervalue(3,reglevel)
78      if (level + amount) <= MAX_LEVEL then
79          ' Pump water into next drum
80          SetRegisterValue 3, reglevel, level+amount
81      else
82          ' Pump water into next drum and some water overflows
83          SetRegisterValue 3, reglevel, MAX_LEVEL
84          overflowed = getregistervalue(3,50) + ((level + amount) - MAX_LEVEL)
85          SetRegisterValue 3, 50, overflowed
86      end if
87  End Sub
88
89  Sub PumpWater(reglevel, nreglevel, amount)
90      Call LeakDrum(reglevel, amount)
91      Call FillDrum(nreglevel, amount)
92  End Sub
93
94  Sub InitPhysicalEnvironment()
95      SetRegisterValue 3, 09, VALVE_CLOSE ' Set in valve to closed
96      SetRegisterValue 3, 19, VALVE_CLOSE ' Set out valve to closed
97
98      SetRegisterValue 3, 00, 55          ' Set drum 1 chemical level
99      SetRegisterValue 3, 10, 60          ' Set drum 2 chemical level
100     SetRegisterValue 3, 20, 70          ' Set drum 3 chemical level
101     SetRegisterValue 3, 30, 45          ' Set drum 4 chemical level
102
103     SetRegisterValue 3, 05, PUMP_AUTO   ' Set pump 1 state to auto
104     SetRegisterValue 3, 15, PUMP_OFF    ' Set pump 2 state to off
105     SetRegisterValue 3, 25, PUMP_AUTO   ' Set pump 3 state to auto
106     SetRegisterValue 3, 35, PUMP_OFF    ' Set pump 4 state to off
107
108     SetRegisterValue 3, 50, 0           ' Set overflowed to 0
109 End Sub
```

Appendix D

# Bibliography


[1] Richard Adhikari. As middle eastern malware goes, shamoon's a strange bird. *TechNewsWorld*, 08 2012.

[2] Saurabh Amin, Xavier Litrico, S Shankar Sastry, and Alexandre M Bayen. Stealthy deception attacks on water scada systems. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 161–170. ACM, 2010.

[3] Andrea Carcano, Igor Nai Fovino, Marcelo Masera, and Alberto Trombetta. Scada malware, a proof of concept. In *Critical Information Infrastructure Security*, pages 211–222. Springer, 2009.

[4] Alvaro A Cárdenas, Saurabh Amin, Zong-Syun Lin, Yu-Lun Huang, Chi-Yen Huang, and Shankar Sastry. Attacks against process control systems: risk assessment, detection, and response. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 355–366. ACM, 2011.

[5] Wang Chunlei, Fang Lan, and Dai Yiqi. A simulation environment for scada security analysis and assessment. In *Measuring Technology and Mechatronics Automation (ICMTMA), 2010 International Conference on*, volume 1, pages 342–347. IEEE, 2010.

[6] Cisco. *Ethernet-to-the-Factory 1.2 Design and Implementation Guide*. Cisco, 1.2 edition, July 22 2008. Text Part Number: OL-14268-01.

[7] Budapest University of Technology CrySyS Lab and Economics. skywiper (a.k.a. flame a.k.a. flamer) : A complex malware for targeted attacks. page 64, 05 2012.

[8] Alexander Gostev (Kaspersky Lab Expert). The flame: Questions and answers. *Securelist*, 05 2012.

[9] Nicolas Falliere, Liam O Murchu, and Eric Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[10] I Nai Fovino, Marcelo Masera, Luca Guidi, and Giorgio Carpi. An experimental platform for assessing scada vulnerabilities and countermeasures in power plants. In *Human System Interactions (HSI), 2010 3rd Conference on*, pages 679–686. IEEE, 2010.

[11] Bela Genge, Igor Nai Fovino, Christos Siaterlis, and Marcelo Masera. Analyzing cyber-physical attacks on networked industrial control systems. In *Critical Infrastructure Protection V*, pages 167–183. Springer, 2011.

[12] Béla Genge, Christos Siaterlis, and Marc Hohenadel. Impact of network infrastructure parameters to the effectiveness of cyber attacks against industrial control systems. 2006.

[13] Béla Genge, Christos Siaterlis, Igor Nai Fovino, and Marcelo Masera. A cyber-physical experimentation environment for the security analysis of networked industrial control systems. *Computers & Electrical Engineering*, 2012.

[14] Williams T J. *The Purdue enterprise reference model, a technical guide for CIM planning and implementation*. Instrumentation, System and Automation Society, 1992.

[15] Kaspersky. Resource 207: Kaspersky lab research proves that stuxnet and flame developers are connected. 06 2012.

[16] Karen Scarfone Keith Stouffer, Joe Falco. *Guide to Industrial Control Systems (ICS) Security*. National Institute of Standards and Technology, special publication 800-82 edition, June 2011.

[17] F.L. Lewis. *Introduction to Modern Control Theory: A Brief History of Feedback Control*. Prentice-Hall, 1992. Chapter 1 in: F.L. Lewis, Applied Optimal Control and Estimation.

[18] Marcelo Masera, Igor Nai Fovino, and Rafal Leszczyna. Security assessment of a turbo-gas power plant. In *Critical Infrastructure Protection II*, pages 31–40. Springer, 2009.

[19] Modbus.org. *MODBUS over serial line specification and implementation guide*. Modbus.org, http://www.modbus.org/, v1.0 edition, 02 2002.

[20] Thomas Morris, Rayford Vaughn, and Yoginder Dandass. A testbed for scada control system cybersecurity research and pedagogy. In *proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW'11)*, 2011.

[21] Igor Nai Fovino, Andrea Carcano, Marcelo Masera, and Alberto Trombetta. An experimental investigation of malware attacks on scada systems. *International Journal of Critical Infrastructure Protection*, 2(4):139–145, 2009.

[22] Andres Perez-Garcia, Christos Siaterlis, and Marcelo Masera. Designing repeatable experiments on an emulab testbed. In *Broadband Communications, Networks, and Systems*, pages 28–39. Springer, 2012.

[23] Bradley Reaves and Thomas Morris. An open virtual testbed for industrial control system security research. *International Journal of Information Security*, 11(4):215–229, 2012.

[24] David E. Sanger. Obama order sped up wave of cyberattacks against iran. *The New York Times*, 6 2012.

[25] F.A. (Floris) Schoenmakers. Perspectives on control system security. Master thesis, Delft University of Technology, April 2013. Assessing security risks resulting from contradicting values between Operational and Information Technology.

[26] McAfee Foundstone Professional Services and McAfee Labs. Global energy cyberattacks: "night dragon". retrieved. page 19, 2 2011.

[27] Jill Slay and Michael Miller. *Lessons learned from the maroochy water breach*. Springer, 2007.

[28] Symantec. W32.duqu the precursor to the next stuxnet. 11 2011.

[29] André Teixeira, Daniel Pérez, Henrik Sandberg, and Karl Henrik Johansson. Attack models and scenarios for networked control systems. In *Proceedings of the 1st international conference on High Confidence Networked Systems*, pages 55–64. ACM, 2012.

[30] Jun Wu and Kazukuni Kobara. Comparison of tools and simulators for control system security studies. In *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, pages 45–50. IEEE, 2012.