HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems

by

Efrén Darío López Morales

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2020 by the
Graduate Supervisory Committee:

Gail-Joon Ahn, Co-Chair
Adam Doupe, Co-Chair
Carlos E. Rubio-Medrano

ARIZONA STATE UNIVERSITY

May 2020

ABSTRACT

Utilities infrastructure like the electric grid have been the target of more sophisticated cyberattacks designed to disrupt their operation and create social unrest and economical losses. Just in 2016, a cyberattack targeted the Ukrainian power grid and successfully caused a blackout that affected 225,000 customers.

Industrial Control Systems (ICS) are a critical part of this infrastructure. Honeypots are one of the tools that help us capture attack data to better understand new and existing attack methods and strategies. Honeypots are computer systems purposefully left exposed to be broken into. They do not have any inherent value, instead, their value comes when attackers interact with them. However, *state-of-the-art* honeypots lack sophisticated service simulations required to obtain valuable data. Worst, they cannot adapt while ICS malware keeps evolving and attacks patterns are increasingly more sophisticated.

This work presents HoneyPLC: A Next-Generation Honeypot for ICS. HoneyPLC is, the very first medium-interaction ICS honeypot, and includes advanced service simulation modeled after S7-300 and S7-1200 Siemens PLCs, which are widely used in real-life ICS infrastructures. Additionally, HoneyPLC provides much needed extensibility features to prepare for new attack tactics, e.g., exploiting a new vulnerability found in a new PLC model.

HoneyPLC was deployed both in local and public environments, and tested against well-known reconnaissance tools used by attackers such as Nmap and Shodan's Honeyscore. Results show that HoneyPLC is in fact able to fool both tools with a high level of confidence. Also, HoneyPLC recorded high amounts of interesting ICS interactions from all around the globe, proving not only that attackers are in fact targeting ICS systems, but that HoneyPLC provides a higher level of interaction that effectively deceives them.

DEDICATION

Dedicado a Darío, Lupita y Karla

ACKNOWLEDGMENTS

Thank you Carlos, for giving me the opportunity to do research and join SEFCOM, for having faith in my abilities, for being open to discussing research, academics and life in general. Without your advise I could not have done this work.

Thank you Adam and Dr. Ahn for your support, research advise and invaluable insight, and most of all believing in me. It is such an honor and privilege to have you in my committee.

Thank you Yan and Fish, for being such cool advisors and for always encouraging me to believe it can be done, whatever it is.

I also want to thank my labmates Sukwha, Faris, Mehrnoosh, Arvind, Yeonjung, Jaswant, Bonnie and Matthew for being kind, fun and awesome in general. Doing research at SEFCOM has been without a doubt one of the toughest and most challenging parts of my life. Luckily it has also been one of the most rewarding.

Last but certainly not least, I want to thank you Erandi, for your love, kindness and patience. There is no version of this where I make it without you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Industrial Control Systems (ICS) [1] are heavily used by many industries including public utilities such as electricity, water, telecommunications and transportation, just to name a few. These utilities are part of our daily life and we have become so accustomed to them that we hardly event notice them. Any attack targeted against an utility ICS network like the electric grid, for example, would potentially lead to blackouts in a city or an entire geographical region. Regrettably, this proposition is no longer a fiction. The number of attacks targeting ICS has steadily increased since the infamous Stuxnet malware first showed the world that ICS networks are not safe [2]. Indeed, in 2015 a cyberattack targeting the ICS network successfully took down several distribution stations of the Ukrainian power grid. The ensuing outages left approximately 225,000 people without access to electricity [3].

One of the most important components of ICS networks are Programmable Logic Controllers better known as PLCs [1]. They control mission-critical electrical hardware such as pumps or centrifuges. PLCs were the primary target of the Stuxnet malware as they controlled critical physical processes in a nuclear facility.

In order to better understand cyberattacks against ICS, including PLCs, several honeypots have been proposed [4][5][6][7]. Honeypots can be used by researchers in order to obtain data about new hacker techniques and malware behavior [8]. However, current honeypot implementations for ICS fail to provide the necessary features to capture data on the latest and most sophisticated attack techniques. Another limitation of the current literature is the lack of extensibility features for PLCs and network services. Solving this limitation is important because ICS environments are very heterogeneous in terms of types of devices and the network protocols they use. Yet another limitation is that most of the current implementations are *low-interaction.* This interaction level heavily restricts the value of the data that can be gathered from attacker interactions.

Thesis Statement: *We develop a new medium-interaction honeypot for PLCs by leveraging a combination of existing technologies, e.g., s7comm, http, etc., that adapts and solves the shortcomings of the existing honeypots in the literature, allowing for emerging threats for ICS to be better understood and handled.*

In order to provide support for this Thesis Statement, we present HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems. HoneyPLC is, as far as we know, the very first Medium-Interaction ICS honeypot. HoneyPLC includes TCP/IP Stack, s7comm, HTTP and SNMP advanced simulation and provides extensibility features. Out of the box HoneyPLC supports three different PLCs: Siemens S7-300, S7-1200 and S7-1500. We also implement a ladder logic capture feature that automatically saves malicious ladder logic programs. These programs can later be analyzed to reveal new attacker techniques. This feature too, is unique to our approach.

The principal contributions of this Thesis are:

- We provide a summary of the limitations and shortcomings of existing ICS Honeypots and discuss how they address (or not) emerging malware threats, new ICS technology, e.g., new PLC models and new ICS network protocols.

- We present HoneyPLC, the VERY FIRST medium-interaction honeypot for ICS environments. HoneyPLC solves many of the limitations identified by the summary listed above, it goes a step further by introducing extensibility features and customization flexibility.

- We introduce HoneyPLC's ladder logic capture feature which allows HoneyPLC to collect ladder logic malware specially crafted for ICS. As far as we know, this feature is exclusive to HoneyPLC.

- Finally, we provide experimental evidence that demonstrates HoneyPLC is effective at engaging and deceiving attackers (and the tools they use) specifically targeting ICS infrastructures.

This Thesis is organized as follows: Chapter 2 introduces detailed information about ICS themselves, network protocols for ICS, PLCs, honeypots, ICS-specific malware and projects related to this work. In Chapter 3 presents a review of similar approaches found in the literature. Chapter 4 introduces our problem statement and we highlight the problems this thesis strives to solve. Chapter 5 describes the architecture of our approach and the design ideas that went into it. Chapter 6 illustrates how we implemented our proposed architecture in detail and the technical specifications. Chapter 7 exposes our evaluation description, evaluation environments and our results. Finally, in Chapter 8 we delve into a discussion about how our approach ranks up against current literature and we outline what future research could be undertaken as a result of this thesis. Finally, 9 presents our conclusions.

Chapter 2

BACKGROUND

In this chapter we describe the necessary concepts for the reader to understand Honeypots, ICS, PLCs and malware that is specifically crafted to target ICS systems.

2.1 Industrial Control Systems

All around the world, societies rely on basic utility services that are essential for the continuing stability of their economic, social and political systems; e.g., energy, communications, clean water. In the United States, the Presidential Policy Directive 21 defines sixteen critical infrastructure sectors including communications, manufacturing, emergency services, energy among others [9]. The infrastructure of these sectors heavily depends on Industrial Control Systems (ICS), which is a general term that encompasses several types of control systems, including Supervisory Control and Data Acquisition (SCADA) systems [1], Distributed Control Systems (DCS) [1], and other control system configurations such as Programmable Logic Controllers (PLC) and Remote Terminal Units (RTUs) often found in the industrial sectors and critical infrastructures. Figure 2.1 depicts the place that ICS, SCADA, DCS and PLCs occupy in the context of Cyber-Physical Systems (CPS) and Operational Technology (OT). An ICS is composed of different control devices such as electrical, mechanical or hydraulic and they all work together towards an industrial objective like producing nuclear energy or providing clean water to a city. They may contain several control loops, Human Machine Interfaces, diagnostic and maintenance tools and a plethora of specialized network protocols [1]. Figure 2.2 gives us a detailed look at the different components that make up an ICS and how they interact.

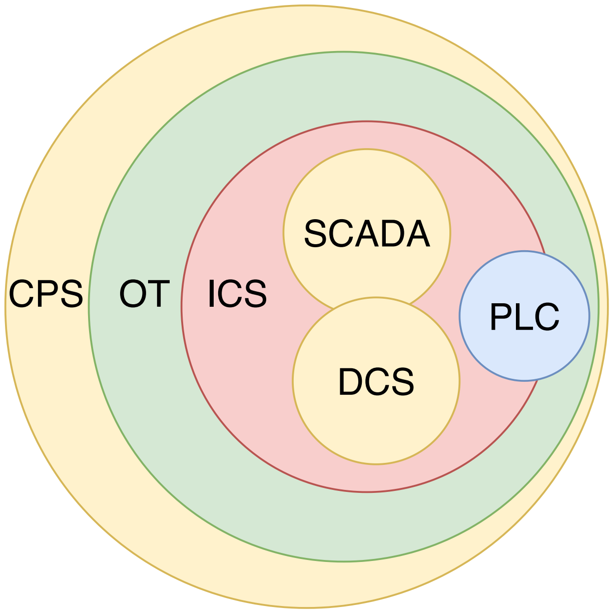


Figure 2.1: The Place of ICS and PLCs in the Big Picture.

The key elements are:

- Human-Machine Interface (HMI): Software and hardware tool that allows human operators to monitor, modify and control the industrial process. They also displays process status and historical reports.
- Actuator: Hydraulic, pneumatic or electrically powered device that positions a valve between the open and closed positions.
- Sensor: Device that that generates a voltage that represents a physical property of a process.
- Controller: Hardware of software that automatically regulates a controlled variable. Programable Logic Controllers s fall in this category, we will expound on them later in this Chapter.
- Remote Diagnostics and Maintenance: Utilities used to prevent, identify and recover from irregular operations and failures.

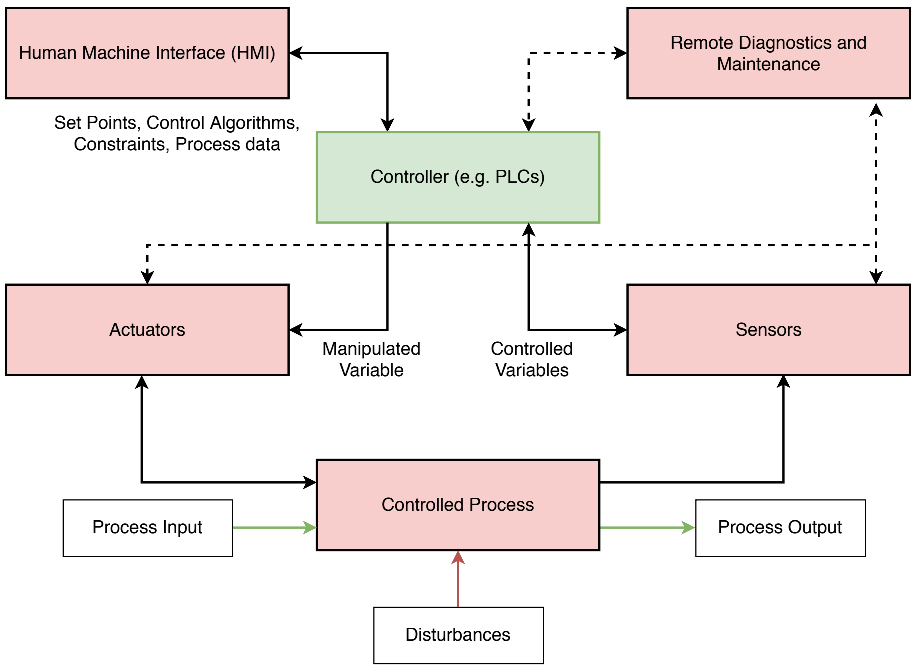


Figure 2.2: Basic Operation of an ICS. Redrawn from original [10].

2.2 Programmable Logic Controllers

A Programmable Logic Controller or PLC for short is a small industrial computer designed to perform logic functions based on input provided by electrical hardware such as pumps, relays, mechanical timers, switches, etc. PLCs have the capability of controlling complex industrial processes and because of this, they are ubiquitous in ICS and SCADA environments [10]. Occasionally PLCs are deployed as field devices like RTUs (special purpose control units designed for remote stations). In these cases, PLCs are referred as RTUs. Some popular PLC manufacturers include Siemens [11], Allen Bradley [12] and ABB [13]. For the remaining of this document, we will focus on Siemens manufactured PLCs, however, our approach can easily work with additional manufacturers.

Internally, PLCs have programmable memory blocks that store instructions to implement different functions, for example, input and output control, counting, logic gates, communication and arithmetic calculation. PLCs memory blocks are used to save data and code. The code can be written in a variety of languages, e.g., STL (Structure List) or STL (Structured Text) [14]. The instruction code is then compiled into MC7 assembly code which is the machine code run by Siemens S7 PLCs [15]. Essentially MC7 code fulfils the same role as Java bytecode. Finally the code is uploaded to the PLC memory via a programming interface connected to a workstation [1][16].

2.3 PLC Memory Blocks

PLCs manufactured by Siemens use specialized memory blocks to store different types of data and make them available to applications like the Siemens Step7 software [16]. In the following sections we describe each type of block in detail.

- Data Blocks (DB): DBs are used to store data that will later be used by a program, different data types can be stored (e.g. Boolean, byte, integer) [17].

- System Data Blocks (SDB): SDBs contain PLC configuration information [16][17]. They may contain PLC model, firmware version, IP address and information about the attached add-ons (e.g. communication processors, frequency converters). These blocks are automatically created and compiled by the PLC and cannot be modified by the user.

- Organization Blocks (OB): OBs are the interfaces between the operating system and the user program [18][19][20]. OBs execute when events occur (e.g. at CPU startup, clocked executions, errors, hardware interrupts). There are several well-known OBs that serve specific roles. For example OB1 in the S7-300 PLC is the default block for cyclic execution of user programs, OB10 to OB17 are time-of-day interrupts and OB35 works as a standard watchdog that runs every 100 ms.

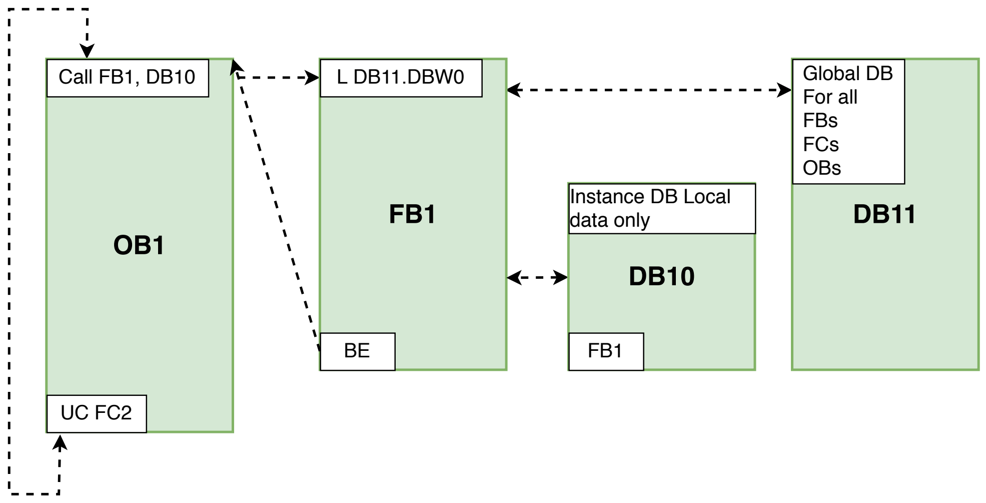


Figure 2.3: Data Block Structure in a LD Program. Redrawn from original [17].

OBs can easily be read and written by an S7 communication protocol enabled application, without the need for authentication or any kind of security. In fact, OB1 and OB35 were the primary targets of the Stuxnet malware payload [16].

- Function Blocks (FB): FBs hold standard code blocks. This code is written in any of the IEC 61131 standard programming languages and is executed by the PLC. Generally FBs are referenced and ran by an OB, for example, OB1 can cyclically run code stored in an arbitrary FB [16][17].

Figure 2.3 depicts an example of the relationships that different types of memory blocks can have.

- OB1: Performs cyclic execution of user program FB 1 using data from DB 10.
- FB 1: Holds user defined code that reads and writes data stored in DB 10 and DB 11.
- DB 10: Stores instance data used by FB 1.
- DB 11: Stores Global data available to all FBs, FCs and OBs.

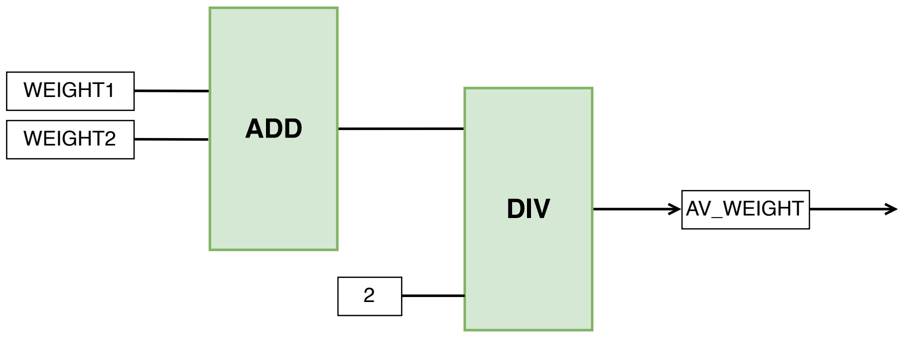


Figure 2.4: Function Block Diagram Example. Redrawn from original [22].

2.4 PLC Programming Languages

PLC devices must be programmed with specialized instructions before being deployed in a production industrial system [21]. The IEC 61131-3:2013 standard specifies the syntax and semantics of a unified suite of programming languages for programmable controllers (PCs). This suite consists of two textual languages, Instruction List (IL) and Structured Text (ST), and two graphical languages, Ladder Diagram (LD) and Function Block Diagram (FBD).

2.4.1 Function Block Diagram

It is described as a graphical programming language that represents signals and data flows through blocks elements that can be later reused. A function block is an instruction unit that takes one or more input values and produces one or more outputs. They are illustrated as a rectangular block with inputs coming in from the left and outputs leaving from the right. A block name is shown inside the block (e.g., the logic gate AND or a division operation DIV) as depicted in Figure 2.4. Function blocks can contain standard functions like logic gates, counters and timers, or custom functions defined by the user, such as custom mathematical equations.

2.4.2 Ladder Logic Diagram

Ladder Diagrams serve as a programming language that uses software diagrams to emulate hardware devices and were originally developed as a transition between old relay control systems to the modern PLCs. They allow industrial technicians and personnel already familiar with old circuit diagrams to easily make small adjustments to modern PLCs and provide maintenance.

The method to create these diagrams involves drag and dropping individual elements into the emulated device as needed depending on the final control process [14] [23]. Elements are patched up and connected to obtain the final design.

2.4.3 Structured Text

Structured text is a programming language that echos the Pascal language [14]. Programs are written in a series of statements separated by semicolons. Statements use predefined functions to change variables that contain stored or I/O data. Like other programming languages, Structured text has analogous operations (e.g. AND, OR, NOT) and statements (e.g. assignment, conditional statements, loop statements). Figure 2.5 shows an structured text piece of code, specifically the if-else syntax and variable assignment.

2.5 ICS Network Protocols

ICS network protocols are used by SCADA systems and PLCs. They are characterized by a lack of standardization and openness [24]. Some of them are proprietary and their specifications are not of public domain.

```
IF(Limit_switch AND Workpiece_Present) THEN
        Gate1 := Open;
        Gate1 := Close;
ELSE
        Gate1 := Close;
        Gate1 := Open;
END_IF
```

Figure 2.5: Structured Text Example. Based on original from [14].

2.5.1 Siemens S7 Communications Protocol

The S7 communications protocol, or s7comm for short [25], is a Siemens proprietary protocol that enables communication between Siemens PLC devices and Siemens Step7 software. It is mainly used for data exchange in PLCs and SCADA systems and for PLC programming. The protocol is transported over TCP using port 102, making it independent of bus medium (e.g. PROFIBUS, Industrial Ethernet). Figure 2.6 describes the s7comm protocol in the context of the network OSI layer model. First, at layers, 5,6 and 7, the S7 application creates a s7comm packet. Second, at layer 4, the s7comm protocol is encapsulated inside a COPT packet. Third, still at layer 5, the COPT packet is encapsulated inside a TPKT packet. Finally, at layer 3, the TPKT packet is encapsulated inside a good ‘ol TCP packet.

The S7 Communication Protocol enables PLC programming and diagnostics and allows applications to read and write data into SFB and FB PLC memory blocks. Because these blocks are used to store industrial environment data and industrial processes instructions, applications should have restricted access to them. However, the S7 communications protocol does not have any kind of security features and any-

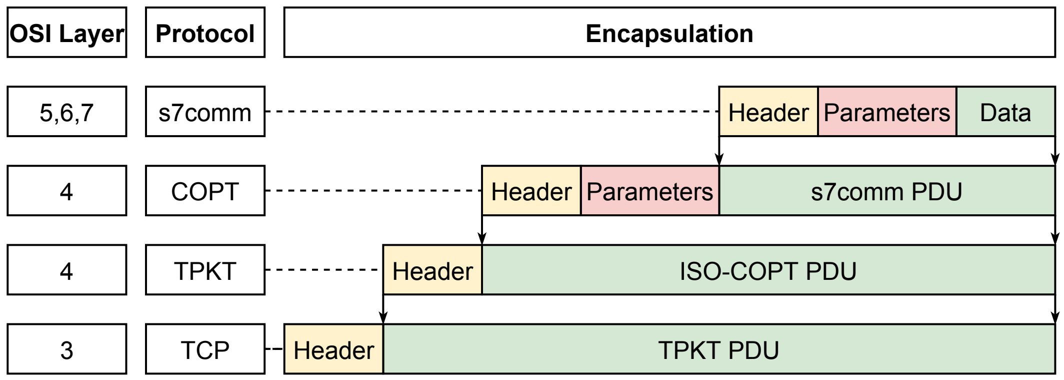


Figure 2.6: S7comm Protocol in the OSI Model. Redrawn from from the original in [26].

body with access to the network can read and write memory block data [27]. Siemens released an enhanced version of the protocol called S7 Plus. This new protocol includes much needed security features such as session ID and encryption. Nevertheless several vulnerabilities have been documented. [28]. We shall refer to this protocol as s7comm in all the remaining chapters of this Thesis.

2.6 Evolution of ICS Malware

For years, different malware with different attack vectors have targeted ICS. In this section we discuss some of the most influential ones and that we discuss later in this document.

2.6.1 Stuxnet

The first ever documented cyber-warfare weapon, Stuxnet, was a turning point in the history of Cybersecurity. It set itself apart from previous malware by showing a high level of sophistication, deep understanding of industrial processes and using not one but four *zero-day* exploits [16].

Stuxnet targeted PLC models 315 and 417 made by Siemens and modified its ladder logic code while concealing itself from ICS administrators [29]. To accomplish this, it first spread itself via USB sticks and the local network, looking for vulnerable Windows workstations. After infecting the Windows workstation it would proceed to infect the Step7 and WinCC Siemens proprietary software by hijacking a DLL (Dynamic-link library) file used to communicate with the PLCs. After careful probing and monitoring the malware would drop its payload only on 315 and 417 models with specific manufacturer numbers and special memory blocks by continuously reading the SDBs as explained in Section 2.3. Specifically, it infected blocks OB1 and OB35 also discussed in Section 2.3. It is believed that Stuxnet successfully sabotaged one of Iran's nuclear facilities [30].

2.6.2 Dragonfly

Also known as Havex malware [31], Dragonfly was a large scale cyberespionage campaign that targeted ICS software in the energy sector in the United States and Europe. In order to infect its targets, three different attack vectors were used. A spam campaign that used spear phishing targeting senior employees in energy companies. Watering Hole attacks [31] that compromised legitimate energy sector websites to redirect the target to another compromised website that hosted the Lightsout exploit which ultimately dropped the Oldrea or Karagany malwares [32] in the target's host. The final attack vector used was *trojanized* software (legitimate software that is turned into malware), the attackers successfully compromised various legitimate ICS software packages and inserted their own malicious code, these trojanized software packages were freely available to download in the vendors official website.

Once a host was infected, the Havex malware leveraged legitimate functionality available through the OPC protocol to draw a map of the industrial devices present in the ICS network. This kind of data would be highly valuable when designing future attacks. Dragonfly was entirely focused on spying and gathering information on ICS networks. However it did not damage any physical infrastructure or provoke any service disruption like Stuxnet did, this is discussed in Section 2.6.1.

2.6.3 Crashoverride

Otherwise known as Industroyer [33], CRASHOVERRIDE is a sophisticated malware designed to disrupt ICS networks used in electrical substations. It shows in-depth knowledge of ICS protocols used in electrical industry that would only be possible with access to specialized industrial equipment. CRASHOVERRIDE dealt physical damage by opening the circuit breakers on RTUs (see Chapter 2.2) and keeping them open even if the grid operators tried to close them back to restore the system [2]. It is believed to have been the cause of the power outage in Ukraine in December of 2016.

2.7 Honeypots

Honeypots are computer systems that purposefully expose a set of vulnerabilities and services that can be probed, analyzed and ultimately exploited by any attacker [34]. Although honeypots do not posses any intrinsic value, when a malicious party launches an attack on any of its services, all possible interaction data is monitored, logged and stored for future analysis. This data is highly valuable as it can reveal previously unknown attack methods used by attackers, capture new malware behavior and payload and trigger alerts that a malicious entity is trying to exploit systems in a government, enterprise or university network.

2.8 Levels of Interaction

Depending on the level of interaction that they provide to an attacker, honeypots are commonly categorized as *low-interaction* and *high-interaction*, with a third category where more much research is needed: *medium-interaction* honeypots [8].

2.8.1 Low-Interaction Honeypots

Low-interaction honeypots offer the least amount of functionality to an attacker [8] [34]. The services exposed by this kind of honeypot are usually implemented using simple scripts and finite state machines, or, at the very least, simulate the TCP/IP stack. Low-interaction honeypots have several disadvantages. Because of their limited interaction attackers may not be able to complete their attack steps or even worst, an attacker might realize that she is targeting a fake system. It might seem that these honeypots are not worth the time, however, they posses some advantages over the other kinds. Low-interaction honeypots cannot be fully compromised as they are not real systems, this greatly reduces maintenance costs and time invested in configuration and deployment. Gaspot [35], further described in Chapter 3, is a perfect example of a low-interaction honeypot because it is implemented using a Python script with minimal configuration options.

2.8.2 Medium-Interaction Honeypots

Medium-interaction honeypots provide a middle ground between low and high interaction levels. These honeypots are able to provide a more sophisticated deception to attackers that would otherwise not be possible in a low-interaction honeypot [8] [36]. They still have limitations compared to high-interaction honeypots. To strike a balance, advanced simulations of services are implemented and exposed. Attackers are able to launch more complex attacks, thus increasing the value of the data collected. Nevertheless, the probability of being compromised completely greatly decreases as an operating system is not exposed. Kippo is medium-interaction SSH honeypot [37]. It is written in Python, however, it provides a more refined simulation like creating a fake file system, ability for attackers to cat command files and save files downloaded using wget. These features are superior to a low-interaction implementation but not as sophisticated as an actual SSH server. There are many medium-interaction honeypots out there, however, at the time of writing this Thesis, there are no ICS medium-interaction honeypots.

2.8.3 High-Interaction Honeypots

High-interaction honeypots lie on the other side of the spectrum, they offer the same level of interaction as a real system would, precisely because they are fully fleshed systems with an installed operating system purposefully exposed [34] [8]. An attacker is able to try any exploit that she might in a production system giving more insight into her methods and intentions. Yet, this comes at a great risk, if the honeypot is fully compromised, adversaries might be able to use it to break into other systems in the same network [38]. Because high-interaction honeypots are actually real systems, a real PLC completely exposed to the internet would make a perfectly good example of a high-interaction honeypot.

2.9 Types of Honeypots

Based on what is their purpose, we classify them as Research and Production honeypots, in this section we discuss each one in detail.

2.9.1 Research Honeypots

The main purpose of research honeypots is learning. Specifically, they are used to gather information about the black hat community [8][39]. This information can be new attack methods, new malware behavior, who is attacking, where are they attacking from and what kind of tools they use. Research honeypots don't provide direct value to a corporation and they are mainly found in universities and research laboratories. Research honeypots are more sophisticated that their production counterparts and they offer more customization. Their downside is a higher difficulty to configure and maintain them.

2.9.2 Production Honeypots

A production honeypot is deployed inside an organization's environment in order to protect their resources and lower risk [8][39]. They differ from Research honeypots mainly because they have far less functionality and are easier to deploy and configure. Additionally production honeypots usually mimic the production network and devices they inhabit. The value production honeypots give to organizations include discovering vulnerabilities, triggering alerts and collecting data that can be later used to better secure the corporate network, among others. Cowrie [40], for example, is a medium-interaction SSH honeypot that can easily be deployed and configured in a production network.

2.10 Related Projects

In this section we examine projects that we leverage to build our approach and that will be discussed at lenght in later chapters.

2.10.1 Honeyd

Honeyd is a low-interaction honeypot framework capable of deploying thousands of hosts [34][41]. Honeyd simulates the TCP/IP Stack of computer systems as well as TCP and UDP services. It provides interaction only at the network level and because of its low-interaction nature, it is highly unlikely that attackers can break into the system.

2.10.2 Nmap

Nmap or "Network Mapper" is an open source network scanning tool [42]. It is used around the world by network administrators to manage and monitor network connected devices. Nmap is able to detect what operating system and services a particular device is running by sending raw IP packets over the network.

2.10.3 Shodan

Shodan is a computer search engine and crawler [43][44]. It looks for and indexes exposed devices across the internet. These devices can be webcams, routers, ICS devices among others. Shodan is web based and boasts a graphical user interface. Additionally, Shodan supports the Shodan API, which can be used by advances users to leverage Shodan's more advanced features, like the Honeyscore. It was lunched in 2009 by John Matherly.

2.10.4 Snap7

Snap7 is an open source project that provides Ethernet communication with Siemens S7 PLCs [26]. Its main features are: native multi-architecture support (32 and 64 bit), Windows, Linux, BSD, Oracle Solaris and Mac OSX support, no configuration or installation needed.

Snap7 has sprung additional projects like #7Sharp, a native port of the original Snap7 in C#, Moka 7, a pure Java port and Settimino an Arduino C port.

Chapter 3

RELATED WORK

In this chapter we look at previous research work done in the area of ICS honeypots.

3.1 SCADA HoneyNet Project

The SCADA HoneyNet Project was the first ever honeypot implementation specifically built for ICS released on 2004 [6][45]. The goal of the project was to find out how feasible it was to develop a software framework capable of simulating ICS devices like PLCs. It is classified as a low-interaction honeypot as the services are partially implemented using Python scripts.

3.2 Conpot

Conpot is a low-interaction ICS honeypot implementation that simulates a particular PLC device [4]. Its default configuration emulates a Siemens S7-200 CPU and it can be modified to include additional profiles by editing an XML file. Conpot supports the s7comm, Modbus, HTTP and SNMP protocols. In addition Conpot includes an HMI that can be customized by the user by modifying the default template. Conpot also provides templates in the form of XML files.

3.3 CryPLH

CryPLH presents a new and different approach to building a PLC honeypot [5]. This approach includes HTTP, HTTPS, s7comm and SNMP services running on a Linux host that has been modified to accept connections on specific ports. The s7comm protocol is simulated by showing an incorrect password response and the TCP/IP Stack is simulated via the Linux kernel.

3.4 Gaspot

Gaspot is a low interaction honeypot designed to simulate a gas tank gauge [35]. It was written as a Python script and it can be modified to change temperature, tank name, and volume.

3.5 Digital Bond's SCADA Honeynet

Released in 2006, this Honeynet implementation utilizes two separate Linux virtual machines [45]. The first one includes a Honeywall that monitors traffic alongside a proprietary IDS. The second and most interesting virtual machine hosts a simulation of a Modicon Quantum PLC and several exposed services (Modbus, FTP, Telnet, SNMP, HTTP).

3.6 HoneyPhy

HoneyPhy provides the foundations of an extensible framework for physics-aware honeypots [7]. The proposed framework would have three modules: internet interfaces, process models and device models. Additionally a proof-of-concept implementation backs up the framework design. The implementation includes a heating system, a thermostat, DNP3 network communications and a control relay. Finally a mathematical model that simulates the temperature change is presented based on data from empirical observations of an actual physical design.

Chapter 4

PROBLEM STATEMENT

In this chapter we identify the research gaps and limitations that the current literature presents when it comes to ICS Honeypots. As we discussed in Chapter 1, ICS are ubiquitous in the infrastructure of many utilities we use on a daily basis, from electricity to transportation. Any disruption to these systems would result in massive socioeconomic damage. This scenario is no longer a thing of the future, it is a reality. The number of attacks targeted towards ICS has regularly increased since 2010 when Stuxnet became the first known digital weapon [2]. Later, in 2013, the Dragonfly malware campaign successfully spied on several utility companies as discussed in Chapter 2. However, the worst was yet to happen, in 2015, the Ukraine Power Grid was successfully brought down by a cyberattack. This event marked the first time a country's power grid was compromised by a cyberattack [2].

One of the most important components of ICS networks are Programmable Logic Controllers also known as PLCs [1]. They control key hardware such as pumps or centrifuges. Honeypots are an excellent tool that helps us understand cyberattacks against ICS, including PLCs. Several honeypots implementations have been proposed [4][5][6][7]. Researchers utilize ICS honeypots to obtain valuable data about new attacker tactics and malware behavior [8]. However, current ICS honeypot implementations fail to provide the necessary features to capture data on the latest and most sophisticated attack techniques. Specifically, we identified the following limitations:

- **Limited Interaction:** Current approaches mostly provide *low-interaction* (rudimentary and limited functionality) implementations as is discussed further in Chapter 3. This is a serious limitation that stops current approaches from extracting value from adversarial interactions and malware. CRASHOVERRIDE, discussed in Section 2.6.3, leveraged advanced ICS protocol features that would be better simulated in a medium or high interaction honeypot and would not be possible using a low interaction approach. This would result in the loss of highly valuable interaction data. For example, CryPLH [5] implements the s7comm protocol using a Python script that only simulates an incorrect password screen.

- **Limited Covert Operation:** The moment an attacker discovers the true nature of a honeypot is game over, the attacker might stop interacting with it altogether and stop revealing her attack methods. One of the most used tools for network reconnaissance is Nmap. Therefore, honeypots should aim to fool tools like Nmap to maintain their covert operation. Currently, the SCADA HoneyNet Project is the only other approach in the literature that leverages Honeyd to provide a convincing deception to attackers. All other implementations fail to attempt and sometimes even mention this as it will be shown in Chapter 3.

- **Limited Extensibility:** Another limitation of the current literature is the narrow or nonexistent extensibility support for different PLC devices and network services that are used in ICS in practice. According to Table 8.1, only Conpot provides the ability to customise XML files to support additional PLC devices. More to the point, the heterogeneous nature of ICS protocols requires extensibility to support new PLC devices and new network services. Stuxnet, discussed in Section 2.6.1, targeted different kinds of PLCs. CRASHOVERRIDE, discussed in Section 2.6.3, targeted different network services.

- **No Malware Collection:** The highly specialized nature of ICS devices calls forth highly specialized malware techniques to understand them better. Honeypots are a great tool to collect and analyze malware [41]. Actually, there exists non-ICS medium-interaction honeypots that do just that [36]. However, the current literature does not try to solve this problem at all. And the problem is very real. Stuxnet, discussed in Section 2.6.1 injects malicious ladder logic code to the target PLCs and Dragonfly, discussed in Section 2.6.2, injected malicious code into legitimate manufacturer software.

In Chapter 3 we will expound on the specific limitations of the current literature, however, the key takeaway is that none of the current implementations address all of the above limitations. Taking into account the aforementioned problems, it is clear that we a new medium-interaction ICS honeypot should bolster features that alleviate these limitations.

Chapter 5

ARCHITECTURAL DESIGN

In this chapter we illustrate the four core components (along with their subcomponents) of HoneyPLC and how their design addresses important limitations identified in Chapter 4. Overall, HoneyPLC is designed as a medium-interaction honeypot and its components are depicted in Figure 5.1.

The HoneyPLC architecture consists of the following components:

- **Honeyd Framework:** The function of Honeyd is twofold. First, Honeyd's personality engine allows HoneyPLC to provide a highly sophisticated TCP/IP Stack simulation that fools reconnaissance tools like Nmap. When Nmap tries to fingerprint our HoneyPLC host, Honeyd will respond with the appropriate fingerprint information. As we discussed in Chapter 4, keeping a covert operation is important to keep attackers engaged. Second, Honeyd's Subsystem virtualization enables HoneyPLC to provide extensibility of network services by redirecting network traffic to the correct service simulation, e.g., s7comm server. The Subsystem Virtualization communicates with the Network Services component to forward TCP and UDP requests. This component helps alleviate the Limited Extensibility problem discussed in Chapter 4.

- **Network Services:** HoneyPLC provides three network services: s7comm server, SNMP agent, and HTTP server.

  Our s7comm server provides a sophisticated simulation of the Siemens proprietary protocol. When an attacker tries to read or write information to Honey-

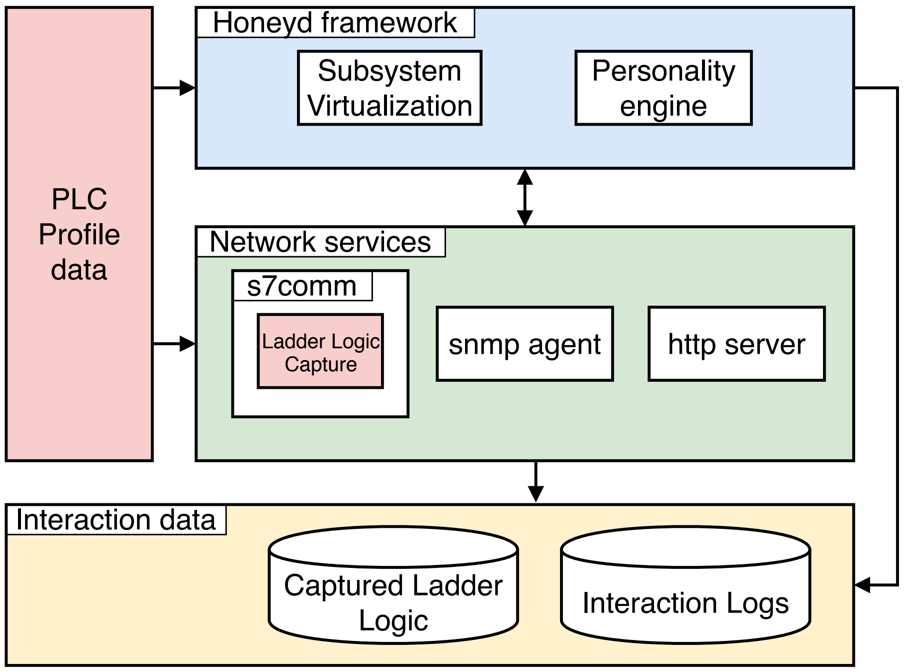


Figure 5.1: The Architecture of HoneyPLC.

PLC, the s7comm server replies to the requests. This server provides a level of interaction not present in low-interaction ICS honeypots. Additionally, our s7comm server includes the Ladder Logic capture feature that writes any ladder logic program that an attacker uploads to HoneyPLC. This feature provides malware collection not present in the literature up until now as disccused in Chapter 3.

The next sub-component is the SNMP agent. Our SNMP agent uses an advanced simulation of the SNMP protocol along with believable MIB data to reply to any SNMP server query. Real PLCs do implement real SNMP agents and having this sub-component adds to the deception capabilities of HoneyPLC.

Finally, the HTTP server provides an advanced simulation of the HTTP server of the Real PLCs and serves websites found in real PLCs.

All three of our Network Services communicate with the Interaction Data component of HoneyPLC to feed their specific log files.

This component is key to addressing the problem found in *low-interaction* ICS honeypots by providing higher interaction features.

- **Interaction Data:** This component holds all of the interaction data gathered by HoneyPLC. It maintains two kinds of data. First, it contains all logs produced by our s7comm servers, the SNMP agent and the HTTP server. Second, it contains all the ladder logic programs that get injected via the s7comm server. This component communicates directly with the Network Services component. It is very important as it holds the much valued data gathered by HoneyPLC.
- **PLC Profile:** The PLC Profile component was designed with the purpose of improving extensibility. It is a collection of PLC Profiles that hold all the required data to simulate a given PLC. The PLC Profile communicates with the Honeyd Framework and Network Services components to customize the PLC that HoneyPLC is simulating at any given time. This component also includes the Profiler tool. We implemented this tool to automate the collection of PLC Profiles from real PLCs. This component addresses the lack of extensibility discussed in Chapter 4.

To hammer this home we present the next example scenario. When we deploy HoneyPLC we must first choose what PLC model we want to simulate, for this example we select the S7-1200 model. We then run HoneyPLC using the S7-1200 Profile. HoneyPLC is running and standing by. Now we suppose that an attacker tries to fingerprint our honeypot with Nmap. All the TCP/IP requests will be handled by Honeyd's personality engine. Because we selected the S7-1200 PLC model in the beginning, Honeyd will use the appropriate fingerprint to reply to Nmap. At this point, Nmap confirms to the attacker that he is dealing with a PLC and not a honeypot. Next, he might try to initiate an s7comm connection to check what PLC memory blocks are available. The connection is first handled by Honeyd and forwarded to the s7comm server. The s7comm server then replies with the information and Honeyd again forwards the replies to the attacker. In the maintime, our s7comm server is logging all the interactions, including source IP and memory block requests. Finally, when the attacker identifies what PLC memory block he wants to attack, he uses an s7comm application like PLCinject to load a malicious ladder logic program. When he does this, the s7comm server will write the program into HoneyPLC filesystem.

At this point, the attacker might do two things. First, continue interacting with HoneyPLC, for example, she might try to download the MIB via the SNMP protocol to get more information about the network configuration or any banner present. Second, she might stop interacting alltogether, at which point HoneyPLC's work is over.

After the attack has been completed we now have very important information logged. We know the public IP address of the attacker, what specific PLC memory blocks the attacker was tergetting, and best of all, we have the critical piece, the ladder logic program she injected. We can use this malware sample to analyze it at the byte level and understand what instructions the attacker wanted the PLC to perform.

Chapter 6

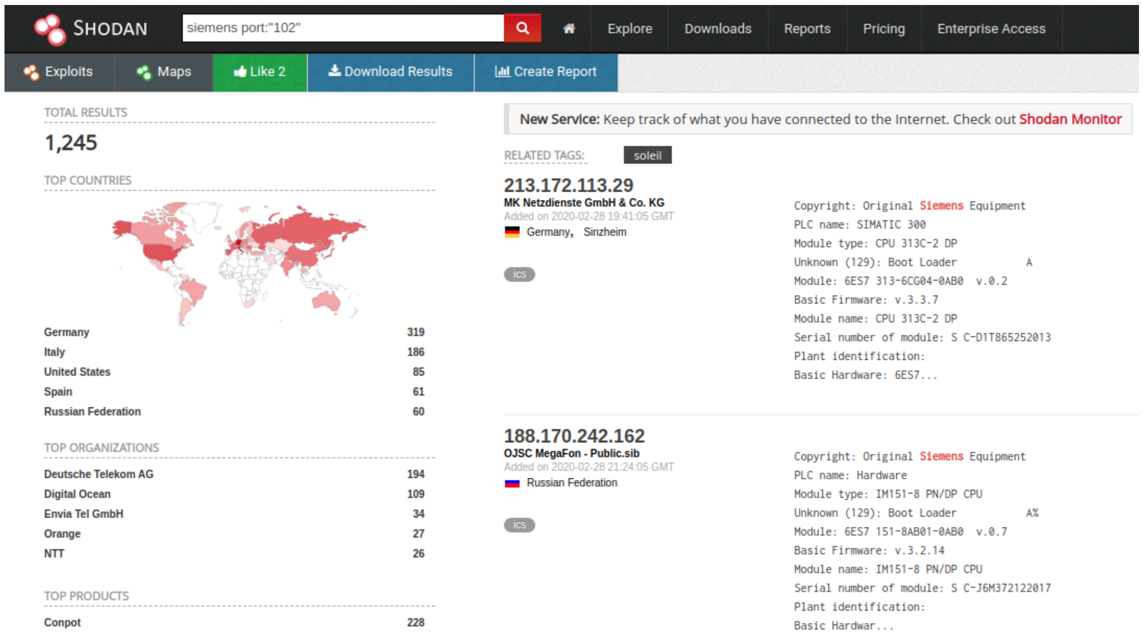
IMPLEMENTATION

In this chapter, we describe how the different parts that compose HoneyPLC are implemented. First, we implement the following network services: s7comm, HTTP and SNMP. The integration of all these elements is made possible by the Honeyd framework. The Honeyd framework allows us to easily integrate new network services and new HoneyPLC profiles. Additionally Honeyd's personality engine is leveraged in order to provide a sophisticated TCP/IP Stack simulation.

We also detail how HoneyPLC Profiles are structured, generated, and used. Our profiles are based on the following Siemens PLC models: S7-300, S7-1200 and S7-1500. We designed our implementation around these particular PLC models as they are commonly found in the wild as shown in Figure 6.1. This figure shows more than a thousand internet-facing Siemens PLCs with the s7comm protocol enabled across different countries.

Although, the list of supported PLC devices might seem limited, the Profiler feature of HoneyPLC makes it possible to extend the number of supported PLCs. In the following sections we describe our implementation approach for the Profiler feature and each of the supported honeypot attack surfaces.

What follows is an in-depth discussion about how each component was implemented and integrated.



New Service: Keep track of what you have connected to the Internet. Check out Shodan Monitor

TOTAL RESULTS

1,245

TOP COUNTRIES

Germany	319
Italy	186
United States	85
Spain	61
Russian Federation	60

TOP ORGANIZATIONS

Deutsche Telekom AG	194
Digital Ocean	109
Envia Tel GmbH	34
Orange	27
NTT	26

TOP PRODUCTS

Conpot	228

RELATED TAGS: soleil

213.172.113.29
MK Netzdienste GmbH & Co. KG
Added on 2020-02-28 19:41:05 GMT
Germany, Sinzheim
ics

```
Copyright: Original Siemens Equipment
PLC name: SIMATIC 300
Module type: CPU 313C-2 DP
Unknown (129): Boot Loader           A
Module: 6ES7 313-6CG04-0AB0  v.0.2
Basic Firmware: v.3.3.7
Module name: CPU 313C-2 DP
Serial number of module: S C-D1T865252013
Plant identification:
Basic Hardware: 6ES7...
```

188.170.242.162
OJSC MegaFon - Public.sib
Added on 2020-02-28 21:24:05 GMT
Russian Federation
ics

```
Copyright: Original Siemens Equipment
PLC name: Hardware
Module type: IM151-8 PN/DP CPU
Unknown (129): Boot Loader           A%
Module: 6ES7 151-8AB01-0AB0  v.0.7
Basic Firmware: v.3.2.14
Module name: IM151-8 PN/DP CPU
Serial number of module: S C-J6M372122017
Plant identification:
Basic Hardwar...
```

Figure 6.1: Shodan Query Revealing Siemens PLCs in the Wild.

6.1 Integration via the Honeyd Framework

In order to build HoneyPLC and integrate our sophisticated service simulations, we leveraged various of Honeyd's capabilities including its personality engine, configuration files and subsystem virtualization.

6.1.1 Honeyd Configuration Files

The first thing that Honeyd requires before running is a configuration file. The configuration file is a simple text file, the language of the file should follow a context free grammar that specifies command options and the correct syntax [34]. Some of the configuration commands allow Honeyd to change the personality of a honeypot, select the network space, set port behavior and more. HoneyPLC leverages the personality setting, subsystem virtualization, port behavior and network space commands. Figure

```
create base
add base subsystem "/usr/share/honeyd/s7commServer" shared restart

clone host1 base
set host1 personality "Siemens Simatic 300 programmable logic controller"

bind 172.16.0.1 host1
```

Figure 6.2: Honeyd Configuration File.

6.2 shows an example of a configuration file. First we create the *base* subsystem and *add* the s7comm server subsystem by providing the required directory. Next we *clone* the *base* to create *host1*. This host will be the actual virtual honeypot that Honeyd exposes. Then we *set* the personality of host1 to use the S7-300 PLC fingerprint. Finally, we *bind* host1 to an IP address.

6.1.2 Honeyd Subsystem Virtualization

The Subsystem Virtualization feature allows for Honeyd to start a regular Unix application within Honeyd's virtual address space [34]. In theory, any Unix application that supports networking can be used as a subsystem by Honeyd and in turn by HoneyPLC.

This features allows the integration of all our network services, s7comm, http and snmp. In Figure 6.2 we can see how the subsystem command is used to start our s7comm server.

6.1.3 TCP/IP Stack Attack Surface

Honeyd's personality engine enables HoneyPLC to deceive fingerprinting tools, e.g., Nmap, and present our honeypot as any device we wish [34]. To achieve this, the engine reads a particular fingerprint from Nmap's database and reverses it. Reversing the fingerprint means that when Honeyd simulates a particular device it introduces its IP/TCP stack peculiarities. These peculiarities are for example, TCP SYN packet flags, IMCP packet flags and timestamps. Figure 6.3 shows an example fingerprint. It contains information about the name of the fingerprint, classification of the device and the results of the multiple tests that Nmap performs for OS detection.

In order to provide the best possible simulation we first used Nmap to generate a detailed TCP/IP Stack fingerprint for each of the Siemens PLCs. We then integrated these fingerprints with the Honeyd fingerprint database. This is done by appending our fingerprints to Honeyd's nmap-os-db text file.

6.2 SNMP Attack Surface

SNMP is used to monitor network connected devices. It listens to requests over UDP port 161. A typical SNMP setup, includes a manager and an agent, the manager continually queries the agent for up-to-date data. A SNMP agent exposes a set of data known as Management Information base or MIB. In order to simulate the SNMP protocol we use a light Python application called *snmpsim*. This application is capable of simulating any SNMP agent based on real time or archived MIB data. When a SNMP request is received by HoneyPLC, snmpsim handles the request and replies with the correct OID in the same way as the real PLC would do.

```
# SIMATIC 1200 PLC
Fingerprint Siemens Simatic 1200 programmable logic controller
Class Siemens | embedded || specialized
SEQ(SP=9E%GCD=1%ISR=A0%TI=I%TS=U)
OPS(O1=M5B4%O2=M578%O3=M280%O4=M5B4%O5=M218%O6=M109)
WIN(W1=2000%W2=2000%W3=2000%W4=2000%W5=2000%W6=2000)
ECN(R=Y%DF=N%TG=20%W=2000%O=M5B4%CC=N%Q=)
T1(R=Y%DF=N%TG=20%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=N%TG=20%W=2000%S=O%A=O%F=A%O=%RD=0%Q=)
T4(R=Y%DF=N%TG=20%W=2000%S=A%A=Z%F=R%O=%RD=0%Q=)
U1(R=N)
IE(R=Y%DFI=N%TG=20%CD=S)
```

Figure 6.3: Nmap Fingerprint of Siemens PLC S7-1200.

6.3 HTTP Attack Surface

Most Siemens PLC devices include an optional HTTP service to manage some of its features. We implemented this with *lighttpd*, a lightweight webserver to handle all HTTP quests. When an HTTP request hits the honeypot, Honeyd's Subsystem virtualization relays it to the lighttpd server and in turn the webserver replies with the website data from a HoneyPLC profile.

6.4 S7comm Attack Surface

The implementation of the s7comm protocol was by far the most challenging aspect of HoneyPLC. At the time of writing, Siemens had not released any specification for their protocol and the information that is available has been collected by third parties like the Snap7 project[26] and the Wireshark Wiki [25].

We leveraged the Snap7 framework described in 2.10.4 to write an s7comm server application in C++. Additionally we modified and recompiled the source code of the main Snap7 library to add our own features. Our s7comm server offers a very sophisticated level of interaction. It simulates a real Siemens PLC and exposes several memory blocks via TCP port 102. Our implementation is so sophisticated that we were able to test it with the Siemens SIMATIC STEP7 software.

One of our new additions to the s7comm server arsenal is the Ladder Logic malware capture. When an adversary uploads a ladder logic program to any of the server's memory blocks this feature automatically writes them into the file Linux file system with the corresponding timestamp. These captured ladder logic programs can then be analyzed at the byte level to extract new attack patterns used by adversaries targeting PLCs.

6.5 HoneyPLC Profiler

The HoneyPLC Profiler was implemented to automate the creation of new HoneyPLC Profiles. It was written as a command line script in Python and interfaces with four different applications: snmpwalk, nmap, and wget. HoneyPLC Profiler requires the IP address of the target PLC device as the only input. It will then run a series of queries to obtain three discrete sets of data from the target PLC: An SNMP MIB, a webiste directory, and an nmap fingerprint.

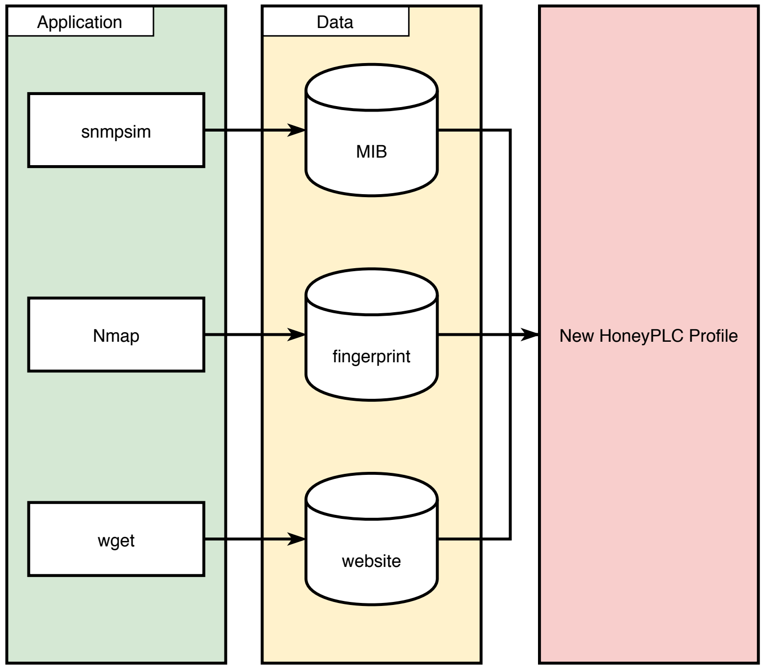


Figure 6.4: HoneyPLC Profiler Architecture.

Figure 6.4 illustrates the HoneyPLC Profiler architecture. First, snmpwalk is used to read all the available OIDs from the public community string. The default configuration assumes the PLC is using SNMP version 1, this is because this is default version of the protocol for the PLCs we have encountered. After all OIDs are read we create an identical MIB to the one used by the PLC. Second, we use Nmap's OS detection to get the TCP/IP stack fingerprint of the target PLC. In order to obtain a better fingerprint we scan all well-known TCP and UDP ports. After the scan is completed we extract the Nmap formatted fingerprint so that Honeyd can understand it. Finally, wget is used to download a complete copy of the administration website of the PLC.

If there are no errors after gathering the data, HoneyPLC Profiler will create a custom directory and data structure that can be used by the HoneyPLC to simulate the target PLC.

6.6 Logging

All of HoneyPLC's components have powerful built-in logging functionalities. We configured Honeyd, lighttpd and snmpsim to automatically log all interactions. Our s7comm server is the exception. The Snap7 framework doesn't create log files automatically modified the main library source code to write to the file system all interactions including: IP address of originating host, timestamp, memory block ID in case of reading or writing. Next, *snmpsim* logs IP information, what OIDs were accessed and timestamps. Finally, the lighttpd webserver includes all the major features of a modern webserver with detailed logging that includes IP address information, accesses website files and timestamps. All of them log every single interaction all the time.

Chapter 7

EVALUATION AND RESULTS

7.1 Covert Operation (Nmap and Honeyscore Experiment)

The moment the honeypot nature of HoneyPLC is revealed to an attacker or malware, the quantity and value of the gathered interaction data drastically decreases. For this reason HoneyPLC has to maintain covertness. In order to evaluate the stealthiness of our approach we decided to test it against two of the most popular tools used by both black hat and white hat community: Nmap and Shodan.

Nmap, as described in Chapter 2.10.2, is a free and open source utility used by millions of people for network discovery and OS fingerprinting. [42]. Shodan's Honeyscore is a tool part of the Shodan API. Its purpose is to expose the many ways in which honeypots fall short of creating realistic simulations. Given an IP address Honeyscore calculates the probability that the host is a honeypot or not. This Honeyscore is a value between 0.0 and 1.0 where 0.0 means that the host is definitively a real system and 1.0 means that the host is definitively a honeypot. In order to learn more about Honeyscore's algorithm we contacted John Matherly and the following are the criteria by which the Honeyscore is calculated. An email (J. Matherly, personal communication, November 27, 2019) gave us specific information about the Honeyscore algorithm and what it looks at:

- Too many open network ports.
- Service not matching the environment, for example, an ICS running on AWS EC2.
- Default settings of known honeypots.

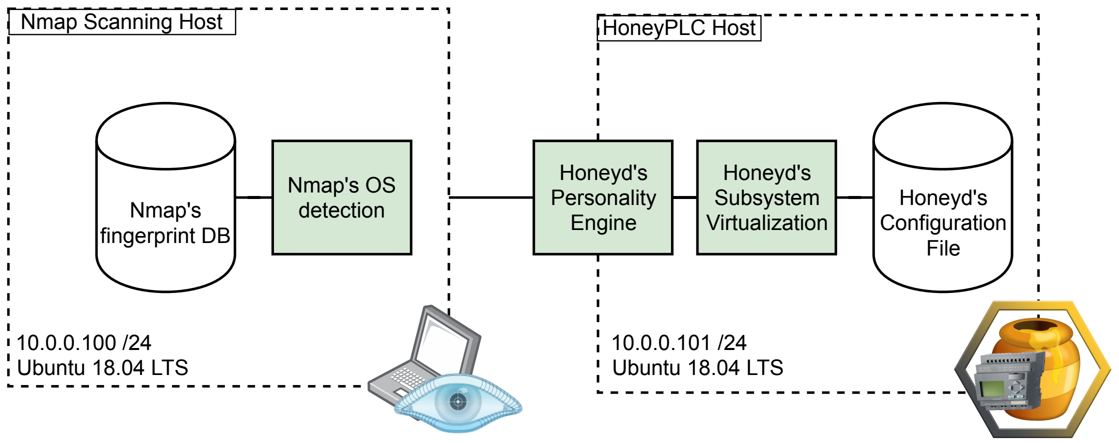


Figure 7.1: Nmap Test Environment.

- If it was a honeypot 1 month ago then it's a honeypot today, even though the configuration looks real.
- Machine learning.
- Same configuration across multiple honeypots.

7.1.1 Nmap Test Environment

Figure 7.1 shows our experimental setup. It was composed of two physical computers. The first one had Ubuntu 18.04 LTS with HoneyPLC and all its components installed. The second one was a simple laptop that had the latest version of Nmap installed. Both hosts were directly connected via an Ethernet cable. Both hosts belonged to the same local network.

In order to conduct a successful experiment we made several configurations to our test environment. First, in the HoneyPLC host we install HoneyPLC and all of its components:

- Honeyd
- lighttpd

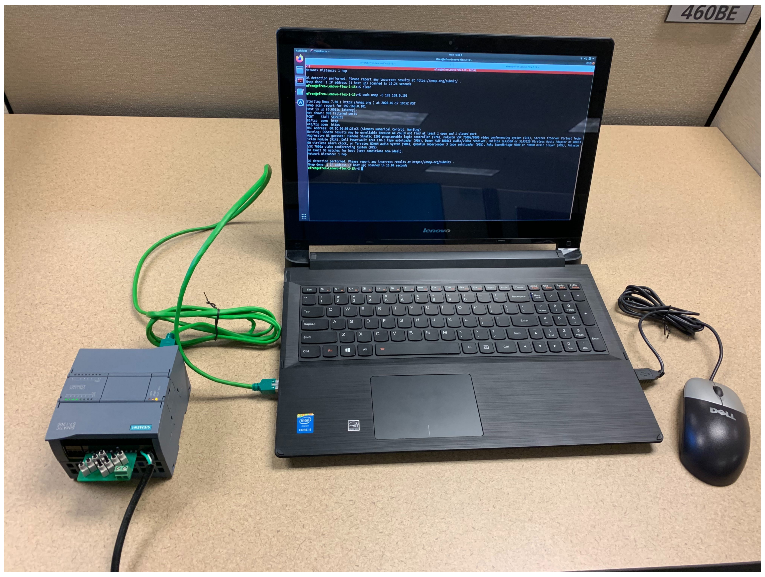


Figure 7.2: Real PLC Nmap Test Environment.

- snmpsim
- s7comm server

We built Honeyd version 1.6d from source, the latest version is available in the official GitHub repository [46]. Next we installed the lighttpd web server version 1.4.45, which is available using Ubuntu's packaging system. After that, we installed snmpsim version 0.4.7 available using Python's package management system pip and all its dependencies. Finally we installed our s7comm server and our custom libsnap7.so library in /usr/lib. Moving on to the laptop used for Nmap scanning we deployed Nmap version 7.80 and proceeded to install our three Siemens PLCs fingerprints in Nmap's fingerprint database nmap-os-db.

Since we needed a baseline to compare the results of our approach, we decided to get the Nmap confidence data of the real PLCs for this purpose. To accomplish this we setup an additional test environment as shown in Figure 7.2. This second test environment was composed of one laptop with Ubuntu 18.04 LTS installed. The laptop also ran Nmap version 7.80 and the Siemens proprietary software S7 Manager. The laptop was directly connected to one of our three different PLCs (S7-300, S7-1200 and S7-1500). Both hosts belong to the same local network and were connected using a standard Ethernet cable.

The PLCs needed to be configured before the laptop could reach them over the network. In order to accomplish this we used the Siemens proprietary software S7 Manager. This software allows us to change the network settings of the PLCs. For our experiments we set the IP addresses of each PLC to 192.168.0.101 with a network mask of 24 bits. Next, we conducted two different sets of Nmap scans with OS detection enabled. One set for HoneyPLC and another set for the real PLCs. Each PLC model is scanned 10 times.

For the HoneyPLC experiment we installed the corresponding HoneyPLC Profile so that the aforementioned applications were correctly configured.

7.1.2 Honeyscore Test Environment

Because Honeyscore works on internet facing IP addresses our Honeyscore test environment needed to be made public. We deployed three EC2 instances accessible from the internet with the following specifications: 2 vCPUs, 4GB RAM and Ubuntu 18.04 LTS OS. Then we deployed HoneyPLC in each one of the servers.

We followed the configuration steps detailed in 7.1.1 to deploy all of our three PLC profiles. Each instance exposes TCP ports 80 and 102 and UDP port 161. We discovered that port 22 for the SSH service was being logged by Shodan, this service should not be present since the real PLCs do not implemented. Because of this, we blocked all connections to SSH, except when originated from our workstations IP addresses, so we could keep administration access.

```
Device type: specialized|printer
Running (JUST GUESSING): Siemens embedded (97%), Brother embedded (90%),
Toshiba embedded (88%)
OS CPE: cpe:/h:siemens:simatic_300 cpe:/h:brother:mfc-7820n
cpe:/h:toshibatec:e-studio-280
Aggressive OS guesses: Siemens Simatic 300 programmable logic controller (97%),
Siemens SPS programmable logic controller (91%), Brother MFC-7820N printer (90%)
```

Figure 7.3: Nmap Scan Results Example.

In order to compare our honeyscores to other real PLCs and other honeypots we decided to leverage Shodan's search engine to gather data of internet facing real PLCs and PLCs flagged as honeypots by Shodan. We looked at open ports, geolocation, honeyscore, PLC model and IP addresses. Then we analyzed these data and use it to compare it to our approach.

7.1.3 Results

The results of our Nmap experiment can be seen in Figure 7.4. Before interpreting our results we should explain how Nmap reports the results of its OS detection scans. After the OS detection scan is completed Nmap can either report a single OS match or report a list of potential OS guesses, each guess with its own confidence rate. Figure 7.3 shows an example.

The results show that for all three PLC models, the Real PLCs get the best confidence by a small margin. However, these results are highly successful since for all scans across all sets Nmap identified the correct PLC model with the highest confidence. Our results are even more impressive when you consider that a Linux

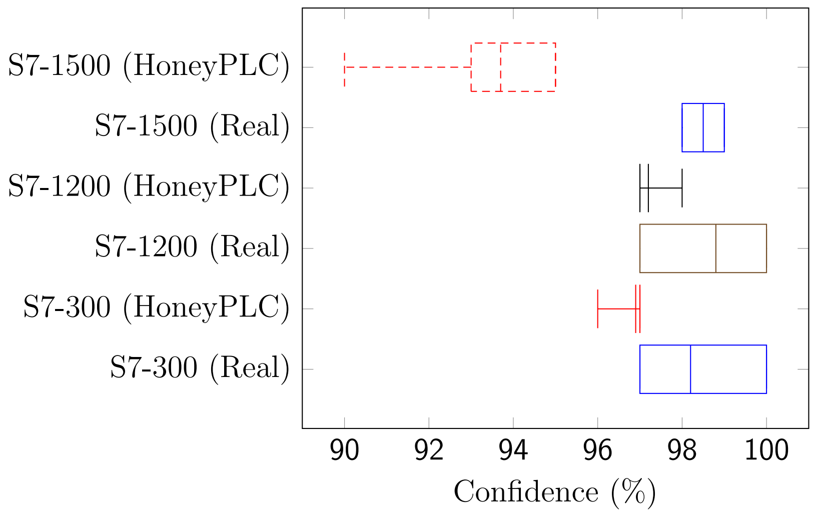


Figure 7.4: Nmap OS Detection Scan Results.

kernel simulation of the TCP/IP Stack will not fool Nmap. In fact, Nmap will report Linux as the OS and not even mention the PLC as one of its guesses [5]. There is a small drop of confidence for the S7-1500 HoneyPLC scans. We believe this change is due the fact that the S7-1500 is much more recent product, it has 2 Ethernet ports instead of 1 and the processor is much faster.

The results of our Shodan experiment are shown in Figure 7.5. It took about a week for Shodan to index our honeypots and identify the s7comm service on port 102 and the http service on port 80. In Figure 7.5 we can observe that Shodan assigns a honeyscore of 0.0 to our S7-300 profile and how this honeyscore compares to real S7-300 PLCs and other S7-300 honeypots found in the wild by Shodan. Next, we can see that our S7-1200 profile get 0.3 and we can compare it to real S7-1200 PLCs indexed by Shodan. We could not find any S7-1200 honeypots using Shodan.

The results of our experiment shows that HoneyPLC is able to fool fingerprint and reconnaissance tools like Nmap, much better than Linux kernel TCP/IP Stack simulation. Additionally we provide detailed evidence data. As far as we know, there has not been a similar evaluation in the literature where confidence data is provided.

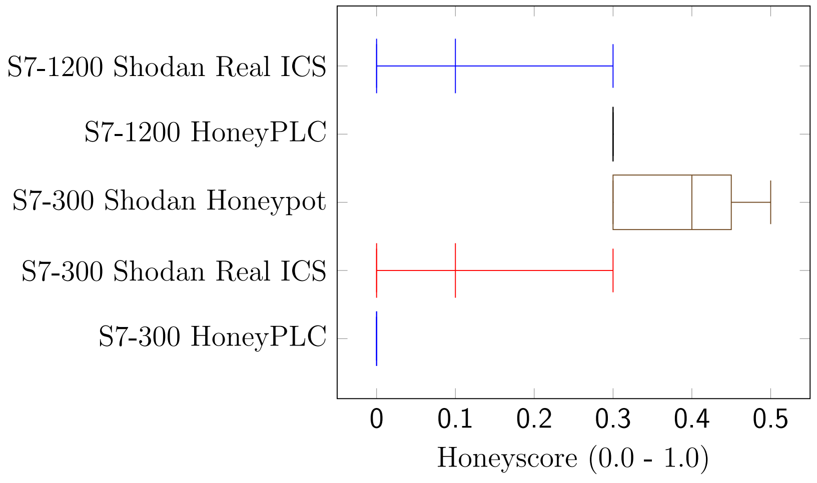


Figure 7.5: Honeyscore Results Comparison.

Additionally, our Honeyscore results show that HoneyPLC performs better than other honeypots found in Shodan.

7.2 Ladder Logic Capture (PLCinject and AWS Experiment)

The goal of this experiment is to demonstrate the capabilities of HoneyPLC's Ladder logic capture feature. To demonstrate the effectiveness of our approach we must answer the question: is it possible for an attacker to successfully inject malicious ladder logic code into HoneyPLC?

In order to try and answer this question we setup a test environment completely exposed to the internet in hopes that an attacker will inject a malicious ladder logic program to HoneyPLC that can later be analyzed to reveal her attack methods. Before doing that we also must make sure that our ladder logic capture feature actually works. To accomplish this goal we use PLCinject. PLCinject is a research tool published by the SCADACS team [47] [48] capable of injecting arbitrary compiled ladder logic programs into a Siemens PLC memory block. Figure 7.6 shows a sample of the ladder logic code dropped by the Stuxnet malware. We also setup an internet facing

```
UC FC1865
POP
L DW#16#DEADF007
==D
BEC
L DW#16#0
L DW#16#0
```

Figure 7.6: Ladder Logic Example.

test environment where we can use PLCinject to test our approach.

7.2.1 Test Environment

For this experiment we use the same AWS test environment described in Section 7.1.2. Additionally we setup a laptop with Ubuntu 18.04 LTS installed and with the latest version of PLCinject available on Github [49]. PLCinject also leverages the Snap7 framework, so we installed our custom library and then compile PLCinject from source. Figure 7.8 illustrates our setup. The PLCinject host contains the ladder logic program sample that PLCinject will upload into HoneyPLC. Figure 7.7 depicts how an injection is performed. The AWS Instance with HoneyPLC installed runs our s7comm server and exposes the standard PLC memory blocks that will be targeted by PLCinject. Finally, all malicious ladder logic programs are saved directly to the Linux file system with its corresponding timestamp.

```
sudo ./plcinject -c 3.15.22.159 -p DB3 -b FC1000 -f /home/edlopezm/PLCinject
plcinject  Copyright (C) 2015  SCADACS (Daniel Marzin, Stephan Lau, Johannes Klick)
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions.
For details check the full license at
http://www.gnu.org/licenses/gpl-3.0.html.

This program is for demonstration purposes only.
You may only use this program to access your own devices and do not
carry out any malicious changes to other devices.
We accept no liability in any way.
Do you agree with that? [y/N]:y
PLC patched!
```

Figure 7.7: PLCinject Execution Example.

7.2.2 Results

Our PLCinject experiment was succesful and we were able to inject a sample ladder logic program into HoneyPLC. After the injection was completed, we logged into our honeypot file system and found the ladder logic file with its corresponding timestamp. Figure 7.9 depicts our experiment result. We use the Linux *ls* command to show the file details.

Additionally, we got a multitude s7comm connections and requests. Our results can be observed in Figure 7.11. The figure shows the percentage distribution of TCP connections made to HoneyPLC simulating a S7-1200. This honeypot received a total of 195140 TCP connections. We can see that the HTTP service received a the

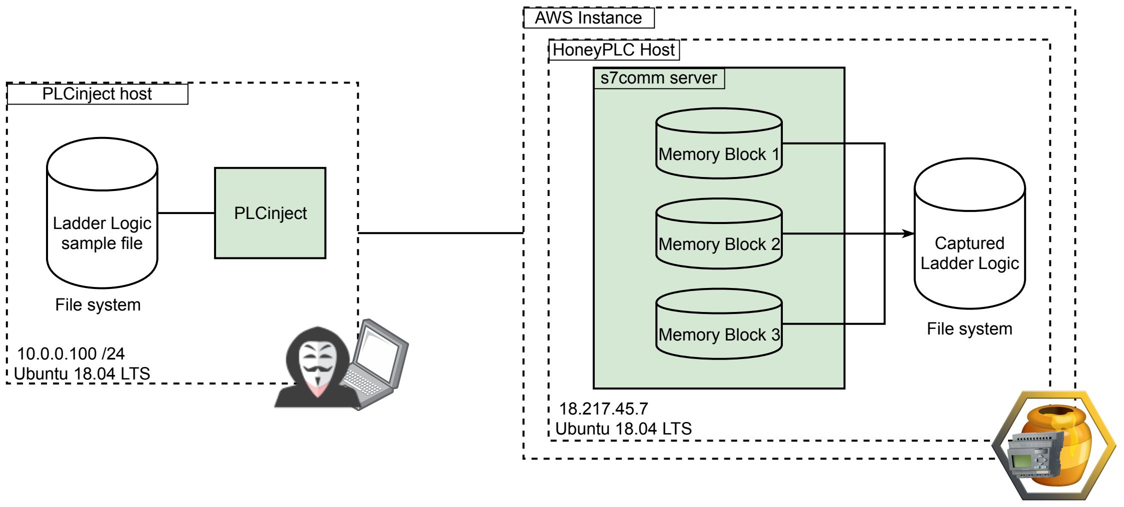


Figure 7.8: PLCinject Test Environment.

```
ubuntu@ip-172-31-9-194:~/s7server$ ls -la
-rw-r--r-- 1 root root 256 Mar 19 21:47 'blockdump_Thu Mar 19 21:47:10 2020'$'\n'
```

Figure 7.9: Ladder Logic Captured in Linux Filesystem.

majority of the connections followed by SSH and s7comm. One thing to note is that most of the SSH connections were started by us in order to administer the honeypot host. The *Other* cathegory includes administrative protocols like NetBIOS and random ports that are not relevant to this work. Finally we are most interested in the s7comm protocol. 7.10 shows how many TCP connections we received and how many read requests for both S7-300 and S7-1200 profiles. Read requests attempt to read information from the PLC memory blocks. The fact that we got these read requests means that attackers interacted with HoneyPLC beyond a simple TCP connection, they might have been performing reconnaissance. At the time of writing this Thesis we did not received any malicious ladder logic. Figure 7.12 shows the *geo-location* of

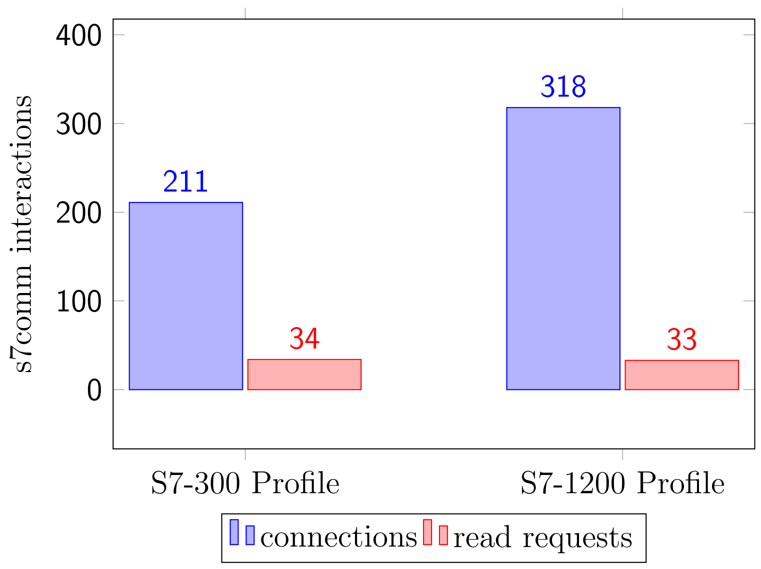


Figure 7.10: Comparison of S7comm Connections Received by HoneyPLC.

the IP addresses that had a TCP conversation using the s7comm protocol. We can clearly see that most of them originated from the United States followed by China and the Netherlands.

Overall, our results clearly show that HoneyPLC provides a higher level of interaction that has not been shown in the current literature and certainly not with our detailed data. HoneyPLC received read requests from s7comm clients from countries usually linked to cyber-warfare that were most likely performing reconnaissance. Additionally, we proved that HoneyPLC is capable of capturing malicious ladder logic programs and saving it for further analysis.

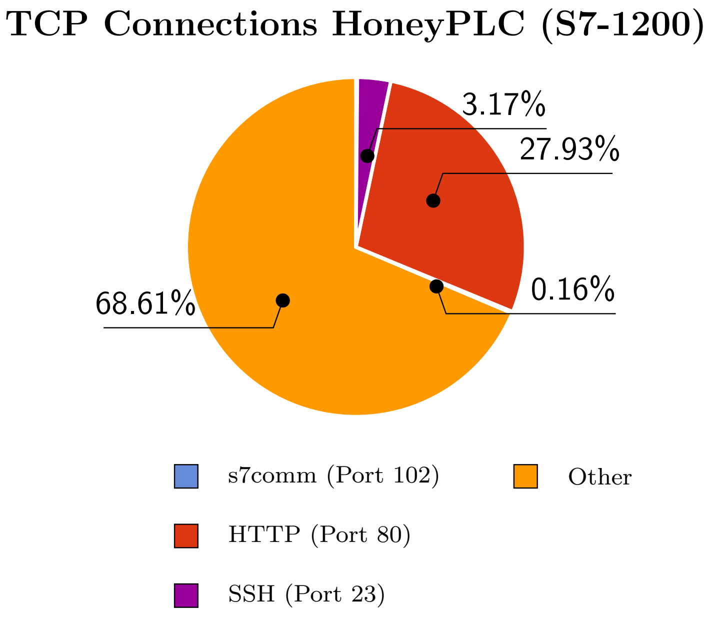


Figure 7.11: Distribution of TCP Connections per Port for HoneyPLC with the S7-1200 Profile.

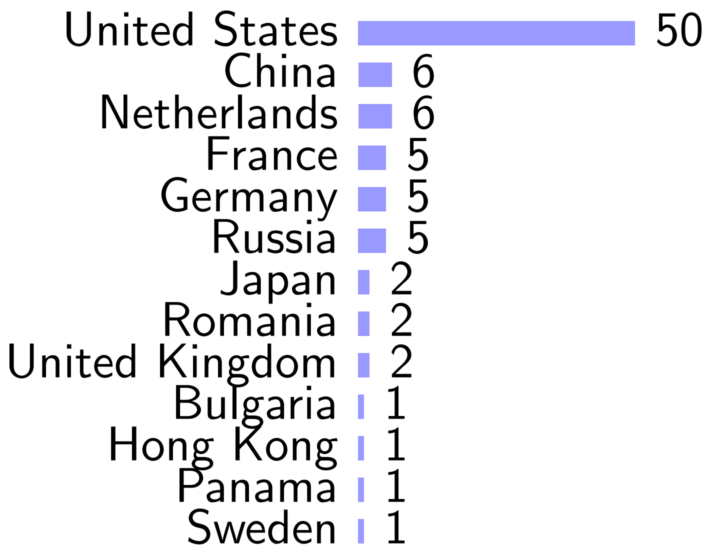


Figure 7.12: S7comm Conversations by Geolocation. These Belong to HoneyPLC Running the S7-1200 Profile.

Chapter 8

DISCUSSION AND FUTURE WORK

HoneyPLC addresses most of the current literature shortcomings. It provides a higher level of interaction than most existing implementations. For example, it simulates the s7comm protocol with PLC memory block read and write features, in contrast, CryPHL [5] simulates the s7comm protocol is simulated by showing an incorrect password response, limiting any other interaction by an attacker. HoneyPLC also improves in extensibility and support by introducing PLC profiles and supporting three PLC devices out of the box. For comparison, Conpot [4] provides XML files to provide extensibility, however, they provide limited data customization and only supports a single PLC device. Regarding, TCP/IP stack simulation, HoneyPLC provides *state-of-the-art* simulation which is superior to CryPLH which uses the Linux kernel to simulate the stack and fails to be identified as a PLC by network scanners.

Even though HoneyPLC introduces several improvements over the current literature still has some limitations. Out of the box it doesn't support additional protocols e.g., Modbus [50], however, this can be updated by future research and using HoneyPLC's extensibility tools. Another limitation is the lack of production ICS environments data, this is not exclusive to HoneyPLC, there is general lack of access to production ICS environments [51].

8.1 Overall Comparison with Existing Approaches

Table 8.1 provides a color-coded comparison of the current literature in ICS honeypots. We now explain what our criteria are for each of the color codes.

- **Red:** The feature has a deficient or non-existent solution coverage.

  For example, in Table 8.1 we have colored Conpot's Network Stack simulation as Red because it fails to provide any solution for this problem. This is a clear contrast with how the SCADA HoneyNet Project handles the Network Stack simulation problem with Honeyd. Or even how Digital Bond's Honeynet tries to solve the problem using the Linux Kernel.

- **Yellow:** The feature has an existent but limited solution coverage.

  For example, in Table 8.1 we colored yellow the honeypots that implement a *low-interaction* level. This is clearly addressing the problem of what interaction level the honeypot implements, however, it is not the best solution when compared to, *medium-interaction* or *high-interaction* solutions.

- **Green:** The feature has an existent and optimal solution coverage.

  For example, HoneyPLC provides PLC profiles to solve the problem of limited extensibility and as a result it is marked as Green. In stark contrast, Gaspot, for example, does not address extensibility at all and this it is marked as Red.

Future work will have to address how we can analyze and extract knowledge from captured ladder logic malware. For example by simulating the execution of malicious code using HoneyPLC itself. There are active research projects to better understand the ladder logic machine code [15]. Another idea would be to beef up HoneyPLC's support by working on more PLC profiles that support additional PLC manufacturers

Approach / Feature	Level of Interaction	Extensibility	Network Stack Simulation	PLC devices	Network Services	Ladder Logic capture	Physics-aware	Logging
SCADA HoneyNet Project [6]	Low	None	Honeyd	Not specified (Generic PLC)	• Modbus • http • ftp • telnet	None	None	Yes
Digital Bond's Honeynet [45]	Low	None	None	Modicom Quantum PLC	• Modbus • http • snmp • telnet	None	None	Yes
CryPLH [5]	High	None	Linux kernel	Siemens S7-300	• s7comm • http • snmp	None	None	Not specified
Conpot [4]	Low	XML template	None	Siemens S7-200	• Modbus • http • snmp	None	None	Yes
Gaspot [35]	Low	None	None	Veeder Root Guardian AST	Not specified	None	None	Yes
HoneyPhy [7]	Hybrid	XML files (not implemented)	None	Generic Analog Thermostat	DNP3	Yes	Not specified	None
SHaPe [52]	Low	ICD or IID file	None	IEC 61850 Compliant PLC (Not Specified)	IEC 61850 (GOOSE)	None	None	Yes
S7commTrace [53]	High	User Template	None	Siemens S7-300	S7comm	None	None	Yes
DiPot [54]	Low	XML template	None	Siemens S7-200	• Modbus • http • snmp • S7comm • Kamstrup	None	None	Yes
Antonioli et al. [55]	High	MiniCPS Configuration	Linux Kernel	Not specified (Generic PLC)	• EtherNet/IP • SSH • Telnet	None	Yes	Yes
HoneyPLC	Medium	PLC Profiles	Honeyd	• S7-300 • S7-1200 • S7-1500	• s7comm • http • snmp	Yes	None	Yes

Table 8.1: Current Literature Comparison.

and models. This would also require updating our Profiler tool to support more network services. Taking HoneyPLC as a proof of concept we could aim for a general model that would simulate different devices found in ICS infrastructure (e.g. MTUs Historian Servers, HMIs).

Chapter 9

CONCLUSION

Attacks targeting ICS are now more real than ever and their consequences are catastrophic. Honeypots help us understand and prepare for these attacks, however, current implementations do not allows us to analyze and tackle brand new threats. In this Thesis we introduced HoneyPLC along with all its components to push ICS honeypots *state-of-the-art* forward. To maintain a covert operation and fool attackers and malware, HoneyPLC implements sophisticated simulations of network services and TCP/IP stack. To keep up with the ever changing malware we introduced the ladder logic capture feature. Additionally we introduce advanced extensibility features, PLC profiles and our Profiler tool, these allow HoneyPLC to adapt to the heterogeneous world of ICS. HoneyPLC might very well be the starting point for future medium-interaction ICS honeypots that help researchers not only to deter ongoing attacks, but to design and evaluate new protection technologies.

REFERENCES

[1] K. Stouffer, S. Lightman, V. Pillitteri, M. Abrams, and A. Hahn, "Nist special publication 800-82, revision 2: Guide to industrial control systems (ics) security," *National Institute of Standards and Technology*, 2014.

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

[3] D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, 2016.

[4] A. Jicha, M. Patton, and H. Chen, "Scada honeypots: An in-depth analysis of conpot," in *2016 IEEE conference on intelligence and security informatics (ISI)*, pp. 196–198, IEEE, 2016.

[5] D. I. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer, "Cryplh: Protecting smart energy systems from targeted attacks with a plc honeypot," in *International Workshop on Smart Grid Security*, pp. 181–192, Springer, 2014.

[6] "SCADA HoneyNet Project: Building Honeypots for Industrial Networks."

[7] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, "Rethinking the honeypot for cyber-physical systems," *IEEE Internet Computing*, vol. 20, no. 5, pp. 9–17, 2016.

[8] I. Mokube and M. Adams, "Honeypots: concepts, approaches, and challenges," in *Proceedings of the 45th annual southeast regional conference*, pp. 321–326, 2007.

[9] P. P. Directive, "Critical infrastructure security and resilience. ppd-21, released february 12, 2013," 2013.

[10] K. Stouffer, J. Falco, and K. Scarfone, "Nist special publication 800-82: Guide to industrial control systems (ics) security," *Gaithersburg, MD: National Institute of Standards and Technology (NIST)*, 2008.

[11] Siemens, "The intelligent choice for your automation task: Simatic controllers." `https://new.siemens.com/global/en/products/automation/systems/industrial/plc.html`. Accessed: 2020-02-24.

[12] Allen-Bradley, "Programmable controllers." `https://ab.rockwellautomation.com/Programmable-Controllers`. Accessed: 2020-02-24.

[13] ABB, "Plc automation." `https://new.abb.com/plc`. Accessed: 2020-02-24.

[14] D. Pessen, "Ladder-diagram design for programmable controllers," *Automatica*, vol. 25, no. 3, pp. 407–412, 1989.

[15] N. Naumenko, "Simatic mc7 code." `https://gitlab.com/nnaumenko/mc7`. Accessed: 2020-02-24.

[16] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[17] S. AG, "Training document for the company-wide automation solution totally integrated automation (tia)." `https://www.automation.siemens.com/sce-static/learning-training-documents/classic/advanced-programming/b04-data-blocks-en.pdf`.

[18] "Which organization blocks can you use in STEP 7 (TIA Portal)? - ID: 40654862 - Industry Support Siemens."

[19] S. AG, "System software for s7-300/400 system and standard functions volume 1/2." `https://cache.industry.siemens.com/dl/files/604/44240604/att_67003/v1/s7sfc_en-EN.pdf`.

[20] C. Lei, L. Donghong, and M. Liang, "The spear to break the security wall of s7commplus," *Blackhat USA*, 2017.

[21] P. Controllers-Part, "3: Programming languages, iec 61131-3: 2013," *International Electrotechnical Commission*, 2013.

[22] W. Bolton, *Programmable logic controllers.* Newnes, 2015.

[23] S. S. Peng and M. C. Zhou, "Ladder diagram and petri-net-based discrete-event control design methods," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 34, no. 4, pp. 523–531, 2004.

[24] Y. Chang, S. Choi, J.-H. Yun, and S. Kim, "One step more: Automatic ics protocol field analysis," in *International Conference on Critical Information Infrastructures Security*, pp. 241–252, Springer, 2017.

[25] "S7comm - The Wireshark Wiki." `https://wiki.wireshark.org/S7comm`, 2016.

[26] D. Nardella, "Snap7." `http://snap7.sourceforge.net/`, 2018.

[27] D. Beresford, "Exploiting siemens simatic s7 plcs," *Black Hat USA*, vol. 16, no. 2, pp. 723–733, 2011.

[28] FortiGuardLabs, "S7.plus.protocol." `https://fortiguard.com/appcontrol/47057`.

[29] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[30] N. Anderson, "Confirmed: Us and israel created stuxnet, lost control of it," *Ars technica*, vol. 1, 2012.

[31] S. I. Response, "Dragonfly: Cyberespionage attacks against energy suppliers," tech. rep., Tech. Rep., July, 2014.

[32] I. Dragos, "Crashoverride: Analysis of the threat to electric grid operations," *Online]: https://dragos. com/blog/crashoverride/CrashOverride-01. pdf*, 2017.

[33] J. Slowik, "Anatomy of an attack: Detecting and defeating crashoverride," *VB2018, October*, 2018.

[34] N. Provos, "Honeyd-a virtual honeypot daemon," in *10th DFN-CERT Workshop, Hamburg, Germany*, vol. 2, p. 4, 2003.

[35] K. Wilhoit and S. Hilt, "The gaspot experiment: Unexamined perils in using,"

[36] G. Wicherski, "Medium interaction honeypots," *German Honeynet Project*, 2006.

[37] U. Tamminen, "Kippo - ssh honeypot." `https://github.com/desaster/kippo`, 2016.

[38] F. Raynal, Y. Berthier, P. Biondi, and D. Kaminsky, "Honeypot forensics," in *Proceedings from the Fifth Annual IEEE SMC Information Assurance Workshop, 2004.*, pp. 22–29, IEEE, 2004.

[39] L. Spitzner, *Honeypots: tracking hackers*, vol. 1. Addison-Wesley Reading, 2003.

[40] M. Oosterhof, "Cowrie honeypot." `https://github.com/cowrie/cowrie`, 2014.

[41] N. Provos and T. Holz, *Virtual honeypots: from botnet tracking to intrusion detection*. Pearson Education, 2007.

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

[43] J. Matherly, "Complete guide to shodan," *Shodan, LLC (2016-02-25)*, vol. 1, 2015.

[44] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.

[45] S. M. Wade, "Scada honeynets: The attractiveness of honeypots as critical infrastructure security tools for the detection and analysis of advanced threats," 2011.

[46] "DataSoft/Honeyd," Feb. 2020. original-date: 2011-12-09T22:40:03Z.

[47] J. Klick and D. Marzin, "Scadacs scadacs," `https://www.scadacs.org/`.

[48] J. Klick, S. Lau, D. Marzin, J.-O. Malchow, and V. Roth, "Internet-facing plcs-a new back orifice," *Blackhat USA*, pp. 22–26, 2015.

[49] "SCADACS/PLCinject," Jan. 2020. original-date: 2015-07-13T09:38:19Z.

[50] A. Swales *et al.*, "Open modbus/tcp specification," *Schneider Electric*, vol. 29, 1999.

[51] E. Byres and J. Lowe, "The myths and facts behind cyber security risks for industrial control systems," in *Proceedings of the VDE Kongress*, vol. 116, pp. 213–218, 2004.

[52] K. Kołtyś and R. Gajewski, "Shape: A honeypot for electric power substation," *Journal of Telecommunications and Information Technology*, no. 4, pp. 37–43, 2015.

[53] F. Xiao, E. Chen, and Q. Xu, "S7commtrace: A high interactive honeypot for industrial control system based on s7 protocol," in *International Conference on Information and Communications Security*, pp. 412–423, Springer, 2017.

[54] J. Cao, W. Li, J. Li, and B. Li, "Dipot: A distributed industrial honeypot system," in *International Conference on Smart Computing and Communication*, pp. 300–309, Springer, 2017.

[55] D. Antonioli, A. Agrawal, and N. O. Tippenhauer, "Towards high-interaction virtual ics honeypots-in-a-box," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, pp. 13–22, 2016.