HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems

Efrén López Morales, Carlos E. Rubio-Medrano, Adam Doupé, Ruoyu Wang, Yan Shoshitaishvili, Tiffany Bao, and Gail-Joon Ahn

1 Introduction

Industrial Control Systems (ICSs) are widely used by many industries including public utilities such as the power grid, water, and telecommunications [48]. These utilities are integral to people's daily life, and any interruption to them may cause significant damage and losses. The increasingly interconnected nature of modern ICS makes them more vulnerable than ever to cyberattacks. For example, a cyberattack that targets a power grid would potentially lead to blackouts in a city or across an entire geographical region. Regrettably, this proposition is no longer a fiction. The number of attacks targeting ICS has been steadily increasing since the infamous Stuxnet malware first showed the world that ICS networks are not secure [14]. Also, in 2015, a cyberattack targeting the Ukrainian power grid successfully took down several of its distribution stations. The ensuing outages left approximately 225,000 people without access to electricity for several hours [7].

E. L. Morales · C. E. Rubio-Medrano (✉)
Texas A&M University—Corpus Christi, Corpus Christi, TX, USA
e-mail: elopezmorales@islander.tamucc.edu; carlos.rubiomedrano@tamucc.edu

A. Doupé · R. Wang · Y. Shoshitaishvili · T. Bao
Arizona State University, Tempe, AZ, USA
e-mail: doupe@asu.edu; fishw@asu.edu; yans@asu.edu; tbao@asu.edu

G.-J. Ahn
Arizona State University, Tempe, AZ, USA
Samsung Research, Seoul, Republic of Korea
e-mail: gahn@asu.edu

T. Bao et al. (eds.), *Cyber Deception*, Advances in Information Security 89,
https://doi.org/10.1007/978-3-031-16613-6_8

1.1 The Problem: Preventing Attacks Targeting ICS via PLCs

One of the key components of ICS networks is Programmable Logic Controllers, better known as PLCs [48]. PLCs are commonly found in supervisory control and data acquisition or SCADA systems. These systems are used to control separated assets that require centralized data acquisition which are a type of ICS [48]. Figure 1 illustrates these relationships. PLCs control mission-critical electrical hardware such as pumps or centrifuges, effectively serving as a bridge between the cyber and the physical worlds. Because of their critical role, PLCs have been recently targeted by cyberattacks, which attempt to disrupt their proper functioning in an effort to affect their corresponding ICS as a whole. As an example, PLCs were the primary target of the Stuxnet malware as they controlled critical physical processes in a nuclear facility. To better understand cyberattacks against ICS and PLCs, several honeypots have been proposed [5, 15, 16, 24, 39, 51]. However, current honeypot implementations for ICS fail to provide the necessary features to capture data for most recent and sophisticated attack techniques. For example, a common limitation exhibited by most of the existing approaches is their low-interaction nature: they usually rely on basic and shallow simulations of network protocols, which usually lack complex functionality that limits the attack vectors and makes them easy to discover by attackers. These shortcomings heavily restrict the value of the attack data that can be gathered by these ICS honeypots.

1.2 Challenges for Solving the Problem

Providing a solution to these issues comes with a set of unique challenges. First, it is difficult to achieve meaningful, step-by-step protocol simulation that can eventually

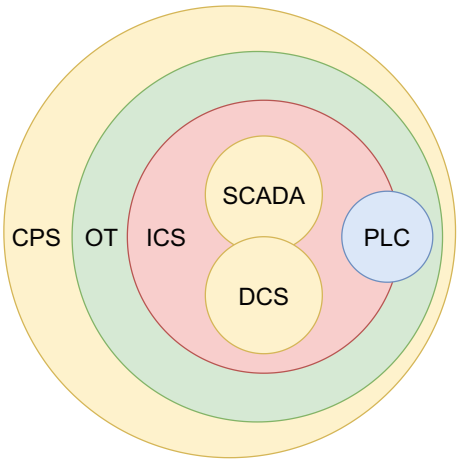

Fig. 1 The relationship between ICS, SCADA, and PLCs, as well as Distributed Control Systems (DCSs) [23], Operational Technology (OT) [13], and Cyber-Physical Systems (CPSs) [44]

result in high-level, deceiving interactions between honeypots and attackers. These *inadequate simulations* complicate concealing the true nature of honeypots up to the point accurate and valuable data, e.g., the actual malicious ladder logic code itself can be retrieved from attackers for further analysis. Second, several network protocols largely used in ICS, e.g., S7comm [51], are *proprietary*, in the sense that no detailed documentation on them is publicly available, which prevents an effective understanding of the protocol, including hidden configuration parameters as well as implicit, undocumented assumptions, which can ultimately reveal the true nature of a honeypot to an attacker. Moreover, existing PLCs used in practice vary in terms of configuration settings, supported protocols, and the way they are customized for different application domains. Creating a general framework that can effectively support such *heterogeneity* of PLCs devices, regardless of their brand and model, without requiring the edition of large and clumsy configuration files, represents a non-trivial challenge.

1.3 Proposed Approach: A Next-Generation Honeypot for ICS

To alleviate the aforementioned concerns targeting ICS worldwide and effectively tackle the research challenges just discussed, this chapter presents HoneyPLC: a high-interaction, extensible, and malware-collecting honeypot modeling PLCs, which is specifically crafted for ICS. HoneyPLC includes *advanced simulations* of the most common network protocols found in PLCs, namely, the TCP/IP Stack, S7comm, HTTP, and SNMP, addressing the challenges introduced by inadequate simulations and protocol closeness as discussed before. As an example, our TCP/IP Stack simulation benefits from the introduction of a novel technique called *fingerprint reversing*, which allows for accurately modeling TCP, ICMP, and UDP probes at runtime, providing an effective, customized response to each interaction as initiated by an attacker, largely increasing the level of engagement and subsequent deception. In addition, our simulation of the S7comm protocol, which is core to PLC communications, provides a level of simulation that is able to trick even proprietary tools such as the Siemens Step7 Manager [4]. Moreover, HoneyPLC also provides *enhanced extensibility features*, allowing for PLCs of different models and manufacturing brands to be effectively simulated, thus addressing the PLC heterogeneity challenge just discussed. We have successfully tested this feature using five *real* PLCs, allowing for HoneyPLC to currently support *out of the box* the Siemens S7-300, S7-1200, and S7-1500, the Allen-Bradley MicroLogix 1100, and the ABB PM554-TP-ETH PLCs. HoneyPLC also implements an advanced simulation of the internal memory blocks featured by modern PLCs, allowing for the *automated capture and storage of malicious ladder logic programs*, which can be later analyzed to reveal new attacking techniques.

The features just discussed are, to the best of our knowledge, exclusive to HoneyPLC and also significantly advance the *state of the art* for ICS honeypots. This positions HoneyPLC as a convenient and flexible tool that can serve as a

reliable basis for the analysis and understanding of emerging threats and attacks, as well as the subsequent development of protection techniques for ICS.

1.4 Contributions to Scientific Literature

Overall, this chapter makes the following contributions:

1. It provides a summary of the limitations and shortcomings of existing ICS Honeypots and discusses how they address (or not) emerging malware threats, as well as new ICS technology, e.g., new PLC models and ICS network protocols.
2. It presents HoneyPLC, a high-interaction honeypot for PLCs, which not only solves many of the limitations of related approaches but also provides convenient support for further understanding and eventually defeating emerging threats for ICS.
3. It introduces the HoneyPLC PLC Profiler Tool, which allows for the effective simulation of many different PLCs regardless of their model and manufacturer.
4. Finally, experimental evidence is provided showing that HoneyPLC is not only effective at engaging and deceiving *state-of-the-art* tools for network reconnaissance but also outperforms existing honeypots in the literature, achieving a performance level comparable to *real* PLC devices.

1.5 Source Code Availability and Chapter Roadmap

In an effort to further open and produce reproducible science, HoneyPLC and all our experimental results are available online.[1] This chapter is an extended version of a paper that appeared at the Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS'20) [25], and it is organized as follows: Sect. 2 introduces detailed information about PLCs, honeypots, ICS-specific malware, as well as similar approaches found in the literature. Section 3 elaborates on the lack of support of such existing approaches for handling emerging threats for ICS, resulting in a problem that is then addressed in Sect. 4. Later, Sect. 5 presents experimental evidence of the suitability of HoneyPLC for being deployed in practice by precisely describing testing environments, procedures, and results. Subsequently, Sect. 6 delves into a discussion about how our approach ranks up against current literature and outlines what future research could be undertaken as a result of this work. Finally, Sect. 7 concludes this chapter.

[1] https://github.com/sefcom/honeyplc.

2 Background and Related Work

Before diving into the details of HoneyPLC, we present some background on the tools and technologies that are addressed in further sections, namely, PLCs themselves, network reconnaissance tools, malware specifically tailored for disrupting ICS, and honeypots that have been developed for protecting ICS environments.

2.1 Programmable Logic Controllers

A Programmable Logic Controller (PLC) 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, making them ubiquitous in ICS and SCADA environments [47]. Some popular PLC manufacturers include Siemens [45], Allen-Bradley [2], and ABB [1]. Internally, PLCs have programmable memory blocks that store instructions to implement different functions, for example, input and output control, counting, logic gates, and arithmetic calculations.

2.2 Network Reconnaissance Tools

In practice, the process of *network reconnaissance* involves identifying the topology of a network, the protocols used, the different devices that may be connected through it, etc. Since such a process is essential for carrying out successful attacks to ICS and PLCs, we now present a set of tools for network reconnaissance that are widely used in practice, which were used to evaluate HoneyPLC as it is discussed in Sects. 3 and 5.

2.2.1 Nmap

Nmap or "Network Mapper" [26] is a popular open-source utility that is able to detect the operating system and services that a particular device is running by sending raw IP packets over the network. Once a given detection scan is completed, Nmap can either report a single OS match or a list of potential OS guesses, each guess with its own confidence percentage rate, in the range of 0 to 100, where 0 denotes the complete absence of confidence and 100 denotes a complete confidence on the projected guess result.

2.2.2 PLCScan

PLCScan [43] is a reconnaissance tool used to scan PLC devices in a given network. PLCScan reveals PLCs that implement the S7comm protocol over TCP port 102 or the Modbus protocol over TCP port 502. It is written as a command line Python script and lists PLC information including basic hardware, serial number, name of the PLC, and firmware version.

2.2.3 Shodan

Shodan is a search engine and crawler [27] specifically tailored for devices exposed across the Internet, e.g., webcams, routers, and ICS devices, among others. The Shodan Honeyscore (part of the Shodan API [27]) is a tool that checks whether a device is a honeypot or not. Given an IP address, the Shodan Honeyscore calculates the probability that the host is a honeypot, in a range between 0.0 and 1.0, where 0.0 means that the host is definitively a *real* system and 1.0 means the host is definitively a honeypot. According to Shodan's creator, the following criteria are used for calculating Honeyscores [28]: (1) too many open network ports, (2) a service not matching the environment, for example, an ICS device running on AWS EC2, (3) known default settings of known honeypots, (4) if a host was initially classified as a honeypot, then it is highly likely that it remains a honeypot today, even though its configuration may look real, (5) a Machine Learning classification algorithm (not disclosed), and, finally, (6) the same configuration being used across multiple honeypots.

2.3 Exemplary ICS Malware

Recently, a series of dedicated malware instances have attempted to disrupt the functioning of ICS environments, and some of them have been successful and have ultimately resulted in costly damages. With that in mind, we now present a summary of the malware that is most relevant to the problem addressed by our proposed HoneyPLC approach.

2.3.1 Stuxnet

The first ever-documented cyber-warfare weapon, Stuxnet, was a turning point in the history of cybersecurity [12], targeting PLC models 315 and 417 made by Siemens to modify their inner ladder logic code while concealing itself from ICS administrators [21]. The malware would first spread itself via USB sticks and the local network, looking for vulnerable Windows workstations. Later, it would proceed to infect the Step7 and WinCC Siemens proprietary software by hijacking

a Dynamic Link Library (DLL) file used to communicate with the PLCs. Finally, the malicious ladder logic payload would be dropped only on the aforementioned models based on specific manufacturer numbers and memory blocks.

2.3.2 Pipedream Toolkit

Pipedream is the seventh documented malware that specifically targets ICS [11]. It is not a single-purpose malware but a modular framework that includes multiple exploits that target different ICS devices. These devices include Open Platform Communications Unified Architecture (OPC UA) servers, Schneider Electric PLCs, and OMRON PLCs. Pipedream is believed to have been developed by a nation state or a state-sponsored group and was classified as an advanced persistent threat or APT by the Department of Energy or DOE [8].

2.3.3 Dragonfly

Also known as Havex malware [37], 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. First, a spam campaign that used spear phishing targeted senior employees in energy companies. Second, Watering Hole attacks [37] that compromised legitimate energy sector websites were deployed to redirect the target to another compromised website that hosted the Lightsout exploit, which ultimately dropped the Oldrea or Karagany malwares [10] in the target's host. The third and final attack vector used was a dedicated *trojanized* software (legitimate software that is turned into malware), the attackers leveraged to successfully compromise various legitimate ICS software packages, ultimately inserted their own malicious code. 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.

2.3.4 Crashoverride

Otherwise known as Industroyer [46], CRASHOVERRIDE is a sophisticated malware designed to disrupt ICS networks used in electrical substations. It shows in-depth knowledge of ICS protocols used in the electrical industry that would only be possible with access to specialized industrial equipment. CRASHOVERRIDE dealt with physical damage by opening circuit breakers and keeping them open even if the grid operators tried to close them back to restore the system. It is believed to have been the cause of the power outage in Ukraine in December of 2016 [14].

2.4 *Honeypots for ICS*

Honeypots are computer systems that purposefully expose a set of vulnerabilities and services that can be probed, analyzed, and ultimately exploited by an attacker [33], allowing for all possible interaction data to be monitored, logged, and stored for future analysis. A summary of existing ICS honeypots is shown in Table 1.

2.4.1 Low-Interaction Honeypots

Low-interaction honeypots offer the least amount of functionality to an attacker [29, 33]. The services exposed by this kind of honeypot are usually implemented using simple scripts and finite state machines. Because of their limited interaction, attackers may not be able to complete their attack steps or may even realize that their target is a fake system. On the other hand, low-interaction honeypots cannot be fully compromised as they are not real systems, which greatly reduces maintenance costs and time invested in configuration and deployment. Gaspot [50] is a low-interaction honeypot written as a Python script that simulates a gas tank gauge. It can be modified to change temperature, tank name, and volume. The SCADA HoneyNet Project was the first honeypot implementation specifically built for ICS [39, 49]. This project was aimed at developing a software framework capable of simulating ICS devices like PLCs using Python scripts. Conpot [16] is also a low-interaction ICS honeypot implementation that simulates a Siemens S7-200 PLC and can be manually modified to simulate other PLCs by editing an XML file.

2.4.2 High-Interaction Honeypots

High-interaction honeypots lie on the other side of the spectrum, as they strive to offer the same level of interaction as a real system [29]. CryPLH is a high-interaction honeypot that simulates an S7-300 Siemens PLC [5] and 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. S7commTrace [51] provides a high-interaction simulation of the S7comm protocol and supports the Siemens S7-300 PLC. Antonioli et al. [3] proposed a high-interaction honeypot that leverages the MiniCPS framework to simulate the Ethernet/IP protocol and a generic PLC. HoneyPhy [24] provides a novel physics-aware model to simulate a generic analog thermostat and the DNP3 protocol.

Table 1 Comparison of existing PLC Honeypots in the literature and HoneyPLC

Keys: ○ = No coverage; ◑ = Limited coverage; ● = Optimal coverage

Approach/ Feature	Extensibility	TCP/IP stack simulation	*Out-of-the-Box* PLCs	ICS network services	Ladder Logic capture	Physics interaction	Logging
Gaspot [50]	○	○	◑	○	○	◑	●
SCADA HoneyNet [39]	○	●	◑	●	○	○	●
Conpot [16]	◑	○	◑	●	○	○	●
Digital Bond's Honeynet [49]	○	○	◑	●	○	○	●
DiPot [6]	◑	○	◑	●	○	○	●
SHaPe [20]	◑	○	◑	◑	○	○	●
CryPLH [5]	○	◑	◑	●	○	○	○
S7commTrace [51]	◑	○	◑	◑	○	○	●
Antonioli et al. [3]	◑	◑	◑	●	○	◑	●
HoneyPhy [24]	◑	○	◑	◑	○	◑	○
HoneyPLC	●	●	●	●	●	○	●
Sections addressing feature	4.2, 5.2	4.3, 5.3, 5.4	4.2, 5.2	4.3, 5.6	4.4, 5.7	6	4.5

3 Limitations of Existing Honeypots

Despite the benefits of honeypots previously discussed, existing honeypots, shown in Table 1, fail to provide the necessary features to capture data on sophisticated attacks, thus exhibiting the following limitations:

L-1 **Limited Extensibility.** A common limitation in the current literature is the narrow extensibility support for the many different PLC devices and network services that are used in ICS in practice and have already been targeted by recent attacks. As an example, Stuxnet and the Kemuri attack targeted different kinds of PLCs, whereas CRASHOVERRIDE targeted different network services, as was discussed in Sect. 2. Following Table 1, several approaches in the literature provide limited extensibility capabilities, which mostly include the manual edition of XML files to support additional PLCs. This process, besides being tedious and time-consuming, may be highly error-prone and may ultimately reveal the true nature of a honeypot to attackers if implemented incorrectly. This is aggravated by the fact most of the approaches in the literature support only one or two PLC models only. In contrast, HoneyPLC currently provides *out-of-the-box* support for 5 PLCs of three major brands, as detailed in Sect. 5.2.

L-2 **Limited Interaction.** Current approaches mostly provide limited functionality when it comes to TCP/IP Stack simulations, as well as native ICS network protocols, as described in Sect. 2. This is a serious limitation that stops current approaches from extracting value from adversarial interactions and malware. As an example, CRASHOVERRIDE leveraged advanced ICS protocol features that are not supported by low-interaction honeypots. This would ultimately result in the loss of highly valuable data. Even high-interaction honeypots fail to provide advanced enough protocol simulations. For example, CryPLH [5] implements the S7comm protocol using a Python script that only simulates an incorrect password screen. HoneyPLC solves this limitation by providing extended support for various networks protocols, as we will discuss in Sect. 4.3 and evaluate through experiments in Sects. 5.3–5.6.

L-3 **Limited Covert Operation.** The moment an attacker discovers the true nature of a honeypot, it is game over, as the attacker might stop interacting with it altogether and stop revealing her attack methods. Therefore, honeypots should aim to fool widely used network reconnaissance tools, e.g., Nmap, introduced in Sect. 2.2, to maintain their covert operation. In such regard, the SCADA HoneyNet Project [39] is the only approach in the literature that provides a convincing deception to attackers. Also, Linux Kernel simulations, implemented by several approaches in the literature, e.g., CryPLH, fail to deceive Nmap. Other work fails to attempt or even mention such a crucial feature. To overcome this, HoneyPLC provides advanced network simulations intended to deceive reconnaissance tools, as shown in Sect. 4.3.

L-4 **No Malware Collection.** The highly specialized nature of ICS devices calls for better analysis, dissection, and understanding techniques specifically tailored

for emerging malware trends. In such regard, honeypots are a great tool to collect and analyze malware [34]. However, as shown in Table 1, there exist no honeypots for ICS in the literature that can provide such functionality. To solve this, HoneyPLC provides a novel feature to capture ladder logic, as described in Sects. 4.4 and 5.7.

4 HoneyPLC: A Convenient High-Interaction Honeypot For PLCs

Having described the limitations of existing approaches, we now present HoneyPLC, an extensible, high-interaction, and malware-collecting honeypot for ICS. HoneyPLC provides advanced protocol simulations, e.g., TCP/IP, S7comm, HTTP, and SNMP, achieving an interaction level comparable to *real* PLCs, ultimately introducing low-to-moderate levels of risk as well as low maintenance costs. We start by providing an illustrative use case scenario, which exemplifies how the different inner modules and components of HoneyPLC interact with an attacker at runtime when an attempt to compromise a PLC is made. Later, we elaborate on how HoneyPLC solves each of the limitations highlighted in Sect. 3.

4.1 Illustrative Use Case Scenario

For illustrative purposes, we present an example use case scenario featuring HoneyPLC, which is based on the architectural design graphically shown in Fig. 2. After this case scenario has been completed, HoneyPLC may have been able to collect crucial information about the attack inside its logging infrastructure: (1) the public IP address of the attacker, (2) the specific PLC memory blocks the attacker was targeting and, best of all, the critical piece, and (3) the ladder logic program he/she has injected. Later on, such a malware sample can be analyzed at the byte level to get a better understanding of the malicious instructions that the attacker wanted the PLC to execute. In Sect. 6, we elaborate on this idea as a part of our future work.

4.1.1 Initial Setup

As it will be further discussed in Sect. 4.2, HoneyPLC can be extended to simulate PLCs of different models, communication protocols, and/or manufacturer brands. With that in mind, the very first step when using HoneyPLC includes choosing the PLC Profile featuring the desired real-life PLC that will be exposed to attackers as a honeypot. This process is shown in Fig. 2 (Step 1). PLC Profiles can be chosen

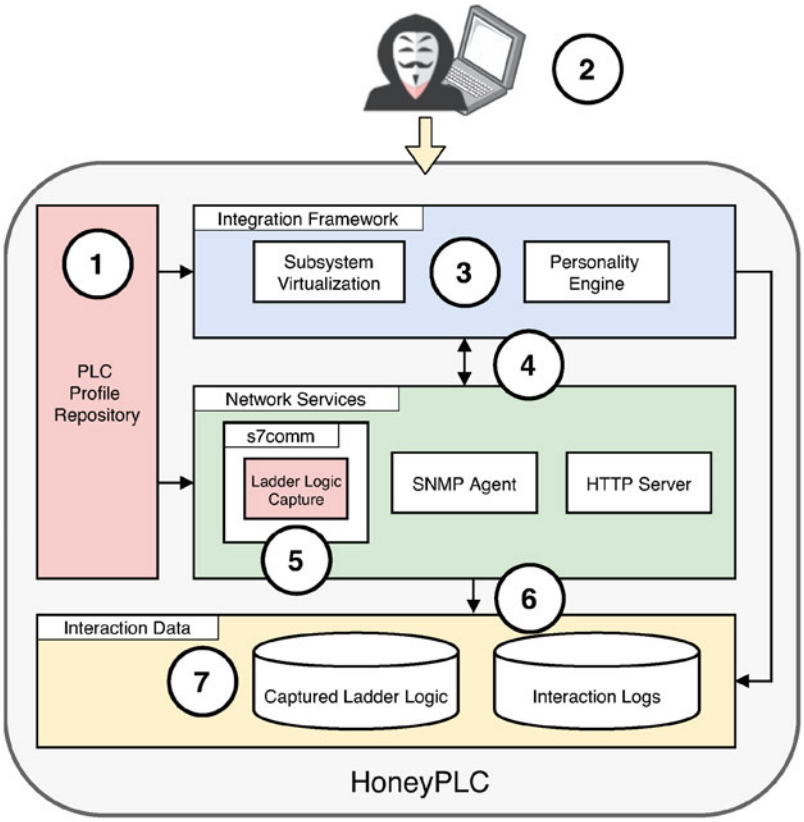

Fig. 2 The architecture of HoneyPLC. Before deployment, a PLC profile is selected from a repository (1). Later, at runtime, an attacker may initiate contact via a dedicated protocol, e.g., S7comm (2). Communications are then processed by the Personality Engine (3), later forwarded to the S7comm server (5), and are eventually logged by the interaction data framework (6). Finally, all code injected by the attacker is captured within the repository module (7)

from a dedicated repository included as a part of HoneyPLC. For the rest of this case scenario, let us assume the S7-1200 model is selected.

4.1.2 Fingerprinting

Once HoneyPLC is deployed, an attacker may try to fingerprint it using a reconnaissance tool such as Nmap or PLCScan (Fig. 2 (Step 2)). When initial contact is established, all the TCP/IP requests will be handled by the HoneyPLC's Personality Engine, which in turn is based on features provided by the Honeyd [9] tool, as it will be further discussed in Sect. 4.3 (Fig. 2 (Step 3)). Since the S7-1200 PLC model was selected in the beginning, the Personality Engine will use the appropriate fingerprint

contained within the PLC Profile to reply to communications started by Nmap. At this point, Nmap may confirm to the attacker that she is dealing with a PLC and not a honeypot, as we show in Sect. 5.

4.1.3 Reconnaissance

In a subsequent step, an attacker might try to initiate an S7comm connection to check what PLC memory blocks are available. As mentioned in Sect. 2, such a process is crucial when attempting to modify the inner ladder logic code of a PLC. The connection is first handled by the HoneyPLC's Network Services module and later forwarded to a dedicated S7comm server (Fig. 2 (Step 4)). The S7comm server then replies with the requested information, and the Integration Framework forwards the replies to the attacker. In the meantime, the S7comm server is logging all the interactions, including the attacker's source IP address and memory block requests made to the PLC.

4.1.4 Code Injection

At this point, when the attacker identifies a PLC memory block suitable for injection, he/she uses an S7comm application like PLCinject [41] to load ladder logic code into the PLC, effectively overwriting any preexisting code and introducing a custom-made malicious payload (Fig. 2 (Step 5)). As a result, the HoneyPLC's S7comm server will write the code into the dedicated HoneyPLC repository, which is managed by the Interaction Data module (Fig. 2 (Steps 6 and 7)).

4.1.5 Confirmation and Farewell

Finally, the attacker has two options. First, he/she can continue interacting with HoneyPLC, e.g., trying 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 altogether, at which point HoneyPLC's work is over.

4.2 Supporting PLC Extensibility

As described in Sect. 3, existing approaches in the literature provide limited support for the large variety of PLC models currently in the market, which limits their suitability for being used in practice. To solve this issue, this section starts by describing how different PLC models are supported by HoneyPLC by means of so-called *PLC Profiles* and then moves on to describe how other models in the market

can be supported by developing new PLC Profiles by means of the HoneyPLC PLC Profiler Tool.

4.2.1 PLC Profiles

The PLC Profile Repository, shown in Fig. 2 (Step 1), is a collection of PLC Profiles that hold all the required data to simulate a given PLC. It communicates with the Integration Framework and Network Services modules to customize the PLC that HoneyPLC is simulating at any given time and addresses the lack of extensibility discussed in Limitation L-1. In turn, a PLC Profile is a collection of three discrete datasets, which allow HoneyPLC to simulate a particular PLC device by means of highly customized simulations of network interactions, as it will be discussed in Sect. 4.3.

- *SNMP MIB.* A Management Information Base (MIB) is a standard used by SNMP agents. Because most PLC devices implement a simple SNMP agent, a custom MIB is needed for HoneyPLC to provide a realistic SNMP simulation.
- *Nmap Fingerprint.* A plain text file with the Nmap fingerprint to effectively simulate the TCP/IP Stack of a particular PLC device. As it will be detailed later in this section, this fingerprint allows HoneyPLC to effectively engage and deceive well-known reconnaissance tools such as Nmap.
- *Management Website.* Some PLC devices provide a light webserver with a splash screen and some configuration options. Because of this, a PLC Profile includes a copy of such website, including, but not limited to, image, HTML, and CSS files.

4.2.2 PLC Profiler Tool

The HoneyPLC Profiler Tool automates the creation of new HoneyPLC Profiles. It interfaces with three different applications: Nmap, (Sect. 2.2), snmpwalk [35], and wget [36]. To obtain the profile for a target PLC, the HoneyPLC Profiler requires the IP address of the PLC device as the only input. Then, the Profiler runs a series of queries to obtain the three discrete sets of data from the target PLC described before: an SNMP MIB, a website directory, and an Nmap fingerprint. First, snmpwalk is used for reading all the available Object IDs (OIDs) from the public community string, creating an identical MIB to the one used by the PLC. OIDs may include, among other important configuration settings, the unique identifier of the PLC, as well as its base IP address. Second, Nmap's OS detection is used to get the TCP/IP stack fingerprint of the target PLC, in a process that includes scanning all well-known TCP and UDP ports. This fingerprint will be later leveraged by HoneyPLC's Integration Framework to provide meaningful TCP/IP interactions as a response to requests initiated by an attacker. Third, wget is used to download a complete copy of the splash screen or administration website, if any. Finally, the HoneyPLC Profiler

will create a custom directory that can be used by HoneyPLC, inside its dedicated PLC Profile Repository, shown in Fig. 2 (1), to simulate the target PLC.

4.3 Supporting Operational Covertness

As described in Sect. 3, being able to engage attackers without revealing a honeypot nature is crucial for obtaining valuable information on the vectors, techniques, and goals being used for compromising PLCs. To this end, this section describes how HoneyPLC supports *meaningful* network interactions leveraging the TCP, IP, S7comm, SNMP, and HTTP protocols, which are widely used by PLCs in practice.

4.3.1 TCP/IP Simulation

Within HoneyPLC's Integration Framework, depicted in Fig. 2, a sophisticated TCP/IP Stack simulation is implemented by leveraging Honeyd [33], a popular framework for honeypot simulation, as well as Nmap, discussed in Sect. 2.2. The process is depicted in Fig. 3. Initially, when a new PLC is to be modeled by HoneyPLC, Nmap is used to generate a detailed TCP/IP Stack fingerprint for it. Next, such a fingerprint is integrated with the Honeyd fingerprint database, by appending it to Honeyd's nmap-os-db text file. Later, at runtime, when a tool

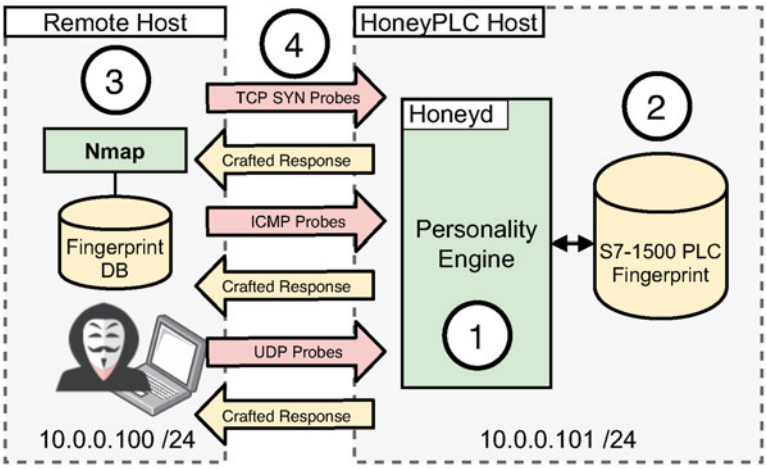

Fig. 3 The HoneyPLC personality engine: first, a PLC Profile is selected from the repository, including its Nmap fingerprint (1). When an attacker tries to fingerprint HoneyPLC using Nmap, such a tool will send a series of Probes to determine the OS or Device (2). HoneyPLC will then reply with appropriately crafted responses that simulate a real PLC, thus effectively deceiving Nmap and the attacker (3)

like Nmap tries to fingerprint a HoneyPLC host, HoneyPLC Personality Engine, leveraging Honeyd, will respond with the appropriate fingerprint information. To achieve this, the Engine reads a particular fingerprint from Nmap's database and *reverses* it, which means that when Honeyd simulates a particular device, it introduces its IP/TCP Stack peculiarities: TCP SYN packet flags, IMCP packet flags, and timestamps. The generation of accurate Nmap fingerprints imposed a variety of challenges. First, PLC devices of different manufacturers and models use different UDP and TCP ports that are not standard or may not be properly defined within the device manuals, e.g., port 2222 for the MicroLogix 1100 PLC. The lack of heterogeneity required us to perform a manual inspection, which was time-consuming and error-prone. Second, we analyzed the Nmap reports that contain the fingerprint results and modified the format to be compatible with the Honeyd fingerprint database. Third, an extensive analysis of the Nmap reports containing the fingerprint results was also required, such that important changes can be introduced for producing better results, i.e., changes in the overall format to make the newly produced fingerprint compatible with the Honeyd fingerprint database. Additionally, the creation of accurate Honeyd templates brought its own set of challenges. For HoneyPLC to provide enhanced interaction capabilities, which can engage attackers for extended periods of time (as we further describe in Sect. 4.3), we significantly improved the standard simulation scripts included within Honeyd. Specifically, we used the subsystem virtualization feature provided by Honeyd: this feature facilitates the integration of the different HoneyPLC components.

4.3.2 S7comm Server

Within HoneyPLC's Network Services Module, depicted in Fig. 2, the S7comm server provides a sophisticated simulation of the Siemens proprietary protocol. It simulates a real Siemens PLC and exposes several memory blocks via TCP port 102. At the time of writing this work, Siemens had not released the specifications of S7comm protocol and the information that is available has been collected by third parties like the Snap7 project [31] and the Wireshark Wiki [38]. We leveraged the Snap7 framework [31, 40] to write an S7comm server application in C++. We modified and recompiled the source code of the main Snap7 library to add our own features. These include logging the S7comm interactions, ladder logic capture, and PLC firmware specifications for all three Siemens PLC models, for example, CPU model, serial number, PLC name label, and copyright among others.

4.3.3 SNMP Server

Within HoneyPLC's Network Services Module, the SNMP Agent implements an advanced simulation of the SNMP protocol along with believable MIB data, effectively allowing HoneyPLC to reply to any external SNMP server query. SNMP is commonly used in practice to monitor network connected devices and listens

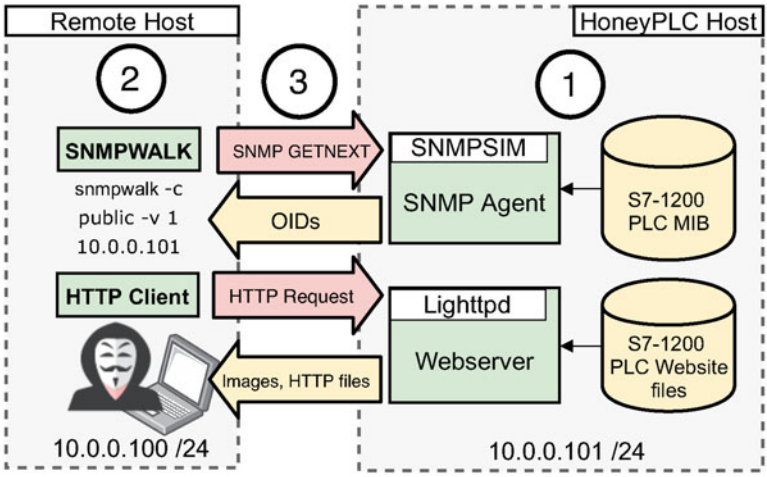

Fig. 4 The HoneyPLC SNMP and the Webserver agents. The MIB database and the website HTTP files, obtained from a PLC profile, are first loaded by each agent (1). Then, the attacker may use SNMPWalk as well as an HTTTP client to establish connection with HoneyPLC (2). Later, each agent will reply to each request using the information obtained from the PLC profile (3)

to requests over UDP port 161. Since *real* PLCs do implement SNMP agents, implementing this sub-component adds to the deception capabilities of HoneyPLC. Our simulation process, shown in Fig. 4 (top), can be described as follows: in practice, a typical SNMP setup includes a *Manager* as well as an *Agent* module. The SNMP Manager continually queries the Agent for up-to-date data. an SNMP Agent exposes a set of data known as Management Information base or MIB. In order to simulate the SNMP protocol, we use the light Python application *snmpsim*, which simulates an SNMP Agent based on real time or archived MIB data. When an SNMP request is received by HoneyPLC, the SNMP Agent replies with an OID as a real PLC would do.

4.3.4 HTTP Server

Finally, the HoneyPLC's HTTP server provides an advanced simulation of the HTTP server of the Real PLCs and serves websites found in real PLCs, as illustrated in Fig. 4 (bottom). As an example, most Siemens PLC devices include an optional HTTP service to manage some of its internal configuration features. This functionality was in turn implemented with lighttpd [19], a lightweight webserver to handle all HTTP quests. When an HTTP request hits HoneyPLC, its Integration Framework relays the request to the lighttpd server. Later, the webserver replies with the website data from a HoneyPLC profile.

4.4 Ladder Logic Collection

HoneyPLC's S7comm Server holds the novel Ladder Logic Capture feature. It writes any ladder logic program that an attacker uploads to HoneyPLC. When an adversary uploads a ladder logic program to any of the S7comm Server memory blocks, while trusting it to be a real PLC, this feature automatically writes them into the file HoneyPLC filesystem with the corresponding timestamp. These captured ladder logic programs can be analyzed at a later stage at the byte level to expose ladder logic instructions and then extract new attack patterns used by adversaries targeting PLCs. We implemented the Ladder Logic Capture component leveraging the Snap7 framework using C++, in a similar fashion as the S7comm Server. Additionally, we modified the Snap7 framework main library files to integrate this feature at the Linux OS level.

4.5 Implementing Record Keeping via Logging

The Interaction Data 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. We configured Honeyd, lighttpd, snmpsim, and the S7comm Server to automatically log all interactions. The S7comm Server writes to the file system all interactions including IP address of originating host, timestamp, and memory block ID in the 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 interaction all the time.

5 Evaluation

As shown throughout Sect. 4, HoneyPLC is designed to effectively deceive attackers into believing that they are dealing with *real* PLCs. This section starts by enumerating a set of experimental questions, which are based on the limitations of existing approaches as presented in Sect. 3. Then we present a series of experiments designed to provide affirmative answers to each question backed up by experimental evidence. For this purpose, we used the following PLC models: Siemens S7-300, S7-1200, and S7-1500, as well as the Allen-Bradley MicroLogix 1100 and the ABB PM554-TP-ETH, which are shown in Fig. 5, as these models are common in

Fig. 5 PLCs procured for experimental purposes including, from left to right, Siemens S7-300, S7-1500, S7-1200, Allen-Bradley MicroLogix 1100, and ABB PM554-TP-ETH

practice. As an example, a query[2] on Shodan [27], shows more than a 1700 Internet-facing PLCs across several different countries. For each experiment, we describe its environmental setup, the methodologies used, and the results obtained. Table 2 shows a summary of the experiments we performed comparing HoneyPLC with other honeypots in the literature whose implementation was either available online or was obtained from their authors upon request. A description of the obtained results is provided next, and an extended discussion comparing HoneyPLC with related work is shown in Sect. 6.

5.1 Experimental Questions

As an initial step, we now enumerate the research questions we have attempted to collect evidence for by means of the experiments shown later in this section. For each question, we describe how it relates to the limitations described in Sect. 3 and what subsections presented later address it.

Q-1 **Can HoneyPLC support different *real* PLCs?**
Since current approaches provided limited support for various types of PLCs being widely used by ICS in practice, we were interested in exploring the capabilities of HoneyPLC to model different PLCs using the PLC Profiler Tool described in Sect. 4.2. This question is related to Limitation L-1, as discussed in Sect. 3. We strive to answer to this question in Sects. 5.2 and 5.2.5.

Q-2 **Can HoneyPLC conceal its honeypot nature from attackers?**

[2] https://www.shodan.io/search?query=siemens+port%3A102.

Table 2 Experimental comparison of PLC Honeypots

Keys: ○ = No coverage; ◑ = Limited coverage; ● = Optimal coverage

Experiment	Conpot [16]	SCADA HoneyNet [39]	Gaspot [50]	S7comm trace [51]	HoneyPLC
Nmap (Sect. 5.3)	◑	●	◑	◑	●
PLCScan (Sect. 5.3)	●	◑	N/A	◑	●
Honeyscore (Sect. 5.4)	●	○	○	○	●
Step 7 Manager (Sect. 5.5)	○	○	N/A	○	●
PLCinject (Sect. 5.7)	○	◑	○	○	●

Table 3 PLC devices supported by ICS Honeypots

Approach	Supported PLC devices
Gaspot [50]	Veeder Root Guardian AST
SCADA HoneyNet [39]	Siemens CP 343-1
Conpot [16]	Siemens S7-200, Allen Bradley LOGIX5561
Digital Bond's Honeynet [49]	Modicon Quantum PLC
DiPot [6]	Siemens S7-200
SHaPe [20]	IEC 61850-Compliant PLC
CryPLH [5]	Siemens S7-300
S7commTrace [51]	Siemens S7-300
Antonioli et al. [3]	Generic PLC
HoneyPhy [24]	Generic Analog Thermostat
HoneyPLC	Siemens S7-300, S7-1200, S7-1500, Allen-Bradley MicroLogix 1100, ABB PM554-TP-ETH

More specifically, can HoneyPLC fool widely used reconnaissance tools? Also, we were interested in obtaining evidence regarding the interactions HoneyPLC may have obtained when deployed in the *wild*, i.e., via an Internet connection. This question is related to Limitations L-2 and L-3. We elaborate on this question in Sects. 5.3, 5.4, and 5.6.

Q-3 **Can HoneyPLC effectively capture Ladder Logic code?**
Since capturing Ladder Logic code represents a highly desirable feature for analyzing threats to ICS, we were interested in exploring the capabilities of HoneyPLC, as described in Sect. 4, to properly carry out such task. This question is related to Limitation L-4 and is addressed in Sect. 5.7.

5.2 Case Study: PLC Profiling

As mentioned in Sect. 3, current state-of-the-art honeypots for PLCs have been modeled over a limited number of PLCs, as shown in Table 3, and support for any extensions is quite limited. Therefore, we were interested in exploring the capabilities of HoneyPLC to support PLCs of different models and manufacturers.

5.2.1 Profiling Siemens PLCs

First, we evaluate the ability of HoneyPLC to support PLCs manufactured by Siemens which are very common both in industry deployments and in academic research [42].

5.2.2 Environment Description

For our first case study, we procured three Siemens PLCs: the S7-300, the S7-1200 and the S7-1500 models, which are shown in Fig. 5. Each PLC was connected to a special power supply and data or Ethernet cables. Additionally, we used the Siemens Step7 Manager, tools to configure IP addressing. We also deployed the HoneyPLC Profiler Tool and Python 3 in a laptop host where we connected our PLCs.

5.2.3 Methodology

We connected each PLC model to our experimental laptop host and used our command line-based HoneyPLC Profiler Tool to create the PLC Profiles for the three PLCs. To launch the tool, we input the PLC IP address and the name of PLC Profile directory. While the HoneyPLC Profiler Tool starts querying data from the PLC progress messages are shown including error messages, if any. We encountered some difficulties while developing and testing the Profiler Tool. First, we had to expand the number of ports scanned to obtain a better Nmap fingerprint, so that Nmap reports it with a higher confidence. We also had to make adjustments to download the PLC websites to include images and correct HTML paths. Also, it was necessary to manually modify the PLC profile HTML files to correct broken links.

5.2.4 Results

Overall, we were successful in creating all three PLC profiles. These profiles were saved in our experimental laptop host file system and were later used in the other experiments depicted in this section. The HoneyPLC Profiler Tool took approximately 5 min to create each profile and we only had to make some small manual modifications to some HTML files, as mentioned before. For PLCs produced by Siemens, the retrieval of their corresponding profiles may be facilitated if the SNMP and the web server services are properly activated beforehand by following the instructions provided by the manufacturer or by using any other S7comm-enabled software, e.g., the Step7 Manager. Failure to perform this step may result in the creation of an incomplete profile.

5.2.5 Profiling Allen-Bradley and ABB PLCs

Additionally, we were interested in exploring the capabilities of HoneyPLC to support PLC manufacturers other than Siemens, so we can provide some general recommendations for practitioners interested in obtaining additional PLC profiles.

5.2.6 Environment Description

For this case study, we procured the Allen-Bradley MicroLogix 1100 and the ABB PM554-TP-ETH PLCs, which are shown in Fig. 5. Additionally, we used Allen-Bradley and ABB software tools to configure their IP addresses.

5.2.7 Methodology

As with our previous case study, we deployed the HoneyPLC Profiler Tool and Python 3 in a laptop host and connected each PLC to a special power supply. Also, we connected each PLC model to our experimental laptop host and used our command line-based HoneyPLC Profiler Tool as before.

5.2.8 Results

We successfully produced a profile for each of the PLCs under analysis and obtained the following recommendations to practitioners. First, for non-Siemens PLCs, it may become necessary to identify the network services they provide, as different vendors may implement a variety of protocols on different ports. As an example, the Allen-Bradley MicroLogix 1100 PLC uses port 80 to implement a light web server, similar to Siemens PLCs, whereas such a feature is not implemented by the ABB PM554-TP-ETH. Second, both non-Siemens PLCs under study also fail to support the SNMP service, which prevents the HoneyPLC Profiler Tool from retrieving a MIB database. Third, the Allen-Bradley MicroLogix 1100 PLC implements the industry standard EtherNet/IP protocol on port 2222 for configuration purposes, which differs from Siemens models that use the proprietary S7comm protocol. These differences may ultimately result in PLC Profiles that are different from the ones obtained for Siemens PLCs and may need to be subsequently addressed on a case-by-case basis. Fourth, whereas the Siemens PLCs use the proprietary S7comm protocol for loading Ladder Logic programs, the Allen-Bradley MicroLogix 1100 uses the Ethernet/IP protocol. In such regard, the ABB PM554-TP-ETH PLC uses the Nucleus Sand Database, which is mostly used for database record keeping, and whose use in PLC devices is not customary. Because both protocols are not currently supported by HoneyPLC, additional modifications may be required. For example, for the M554-TP-ETH PLC Profile, we modified the Honeyd template to open port 1201 as a Nucleus Sand DB simulation that can be used through the subsystem virtualization is not currently supported. For the MicroLogix 1100 PLC Profile, we modified the Profiler Tool port scan range to include not only well-known ports but also registered ports such as port 2222. Finally, Table 3 provides a comparison of the PLC models supported *out of the box* by related honeypots for ICS, which were also shown in Table 1. The positive results obtained in our two case studies give support to answer Q-1 in the affirmative.

5.3 Resilience to Reconnaissance Experiment

The moment the true nature of HoneyPLC (or any other honeypot) is revealed to an attacker, the quantity and value of the gathered interaction data may significantly decrease. Therefore, we aimed to test the resilience of HoneyPLC to Nmap and PLCScan, described in Sect. 2, which are well-known tools for reconnaissance. Additionally, we tested how existing honeypots, namely Gaspot [50], S7commTrace [51], SCADA HoneyNet [39], and Conpot [30], perform in this regard.

5.3.1 Environment Description

Our experimental setup was composed of two physical computers: a *desktop* and a *laptop* host. The desktop host featured Ubuntu 18.04 LTS along with HoneyPLC, as well as the following tools: Honeyd, lighttpd, snmpsim, and S7comm server. We built Honeyd version 1.6d from source; the latest version is available in the official GitHub repository [9]. Also, we installed the lighttpd web server version 1.4.45. Next, we installed snmpsim version 0.4.7 and all its dependencies. Finally, we installed our S7comm server and our custom library. Conversely, the laptop host included the latest version of Nmap 7.80 as well as the three Siemens PLCs fingerprints in Nmap's fingerprint database nmap-os-db that were obtained as a result of the previous experiment. Additionally, we installed the latest version of PLCScan obtained from GitHub [43]. Both hosts were directly connected via an Ethernet cable. Subsequently, we downloaded and deployed the related honeypots mentioned before and connected them to the scanning host so that all of them would be in the local network.

5.3.2 Methodology

To create a baseline to compare the results of our experiments, the Nmap confidence data of the *real* PLCs featured in the previous experiment was obtained. With that in mind, a second test environment was composed of an additional host with Ubuntu 18.04 LTS and Nmap 7.80. Later, the additional host was directly connected to one of the three different PLCs (S7-300, S7-1200, and S7-1500) using an Ethernet cable. We installed the Step7 Manager in order to configure the network settings of the PLCs. Next, two different sets of Nmap scans were conducted with OS detection enabled: one set for HoneyPLC and another set for the *real* PLCs. Each PLC model was scanned 10 times. For the HoneyPLC experiment, the corresponding HoneyPLC Profile was installed so that the aforementioned applications were correctly configured. Next, we used PLCScan to scan each PLC Profile in similar

Fig. 6 Nmap scan results for the S7-300 PLC profile

```
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%)
%\end{verbatim}
```

fashion as the Nmap methodology. Afterwards, we turned to Gaspot, S7commTrace, SCADA HoneyNet, and Conpot. Each honeypot was scanned with Nmap's OS detection enabled 10 times. Finally, we used PLCScan on S7commTrace, SCADA HoneyNet, and Conpot. Gaspot was omitted as it does not support the S7comm protocol.

5.3.3 Results

The results of our Nmap experiment can be seen in Fig. 7 and show that for all three PLC models, the *real* PLCs gets the best confidence by a small margin. However, our PLC Profiles as provided by HoneyPLC were really close behind, thus providing positive evidence that our approach can provide effective covertness, as required by our question Q-2. When Nmap cannot detect a perfect OS match, it suggests near-matches. The match has to be very close for Nmap to do this by default. Nmap will tell you when an imperfect match is printed and display its confidence level (percentage) for each guess [32]. As an example, Fig. 6 shows the Nmap Scan results for our S7-300 PLC Profile. These results are encouraging since for all scans across all sets Nmap identified the correct PLC model with the highest confidence. Our PLCScan experiments were also successful, as we were able to obtain and provide real PLC data using PLCScan against HoneyPLC for all three PLC Profiles. In addition, SCADA HoneyNet was identified as a Siemens CP 343-1 PLC, and however, Gaspot, S7commTrace, and Conpot were fingerprinted as Linux OS with a 100% confidence, with no mention of any PLC device. Regarding PLCScan, Conpot was identified as an S7-200 PLC and SCADA HoneyNet and S7commTrace provided connection information but displayed an empty PLCScan report. Our results are even more significant due to the fact that a Linux kernel simulation of the TCP/IP Stack, as implemented by several related approaches, including Gaspot and Conpot, will not deceive Nmap [5].

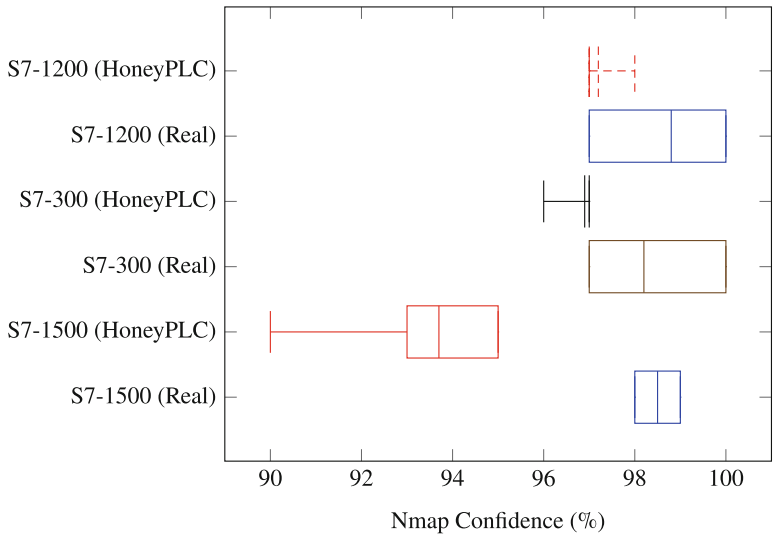

Fig. 7 Nmap scan results. All three profiles obtained at least a 90% confidence rate. The S7-300 and S7-1200 profile obtained rates comparable with their real counterparts. Gaspot and Conpot are fingerprinted as a Linux OS host with a 100% confidence, so they are excluded from this chart

5.4 Shodan's Honeyscore Experiment

As with the previous experiment, Shodan, described in Sect. 2.2, is actively leveraged in practice, along with its corresponding Shodan API to detect honeypots exposed to the Internet with a high degree of accuracy. Therefore, we were interested in the capabilities of HoneyPLC to deal with this state-of-the-art tool.

5.4.1 Environment Description

For this experiment, we deployed three AWS EC2 instances accessible from the Internet with the following specifications: 2 vCPUs, 4GB RAM, and Ubuntu 18.04 LTS OS, exposing TCP ports 80 and 102 and UDP port 161. Then, we deployed HoneyPLC on each one of them featuring all of our three PLC profiles, following the configuration steps detailed in the previous experiment. We also deployed four additional AWS instances hosting Conpot, Gaspot, S7commTrace, and SCADA HoneyNet.

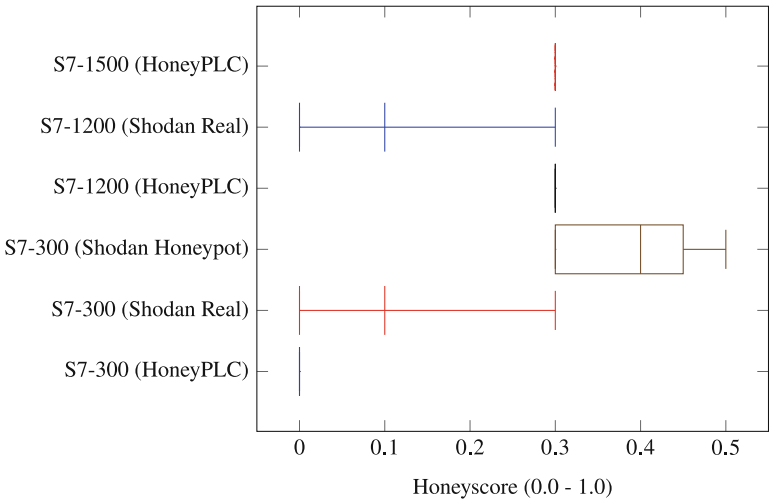

Fig. 8 Shodan Honeyscore results. Our HoneyPLC PLC profiles perform better than other honeypots found in Shodan and at the same level as *real* PLCs

5.4.2 Methodology

We obtained the Shodan Honeyscores, whose methodology is described in Sect. 2.2, of each of our HoneyPLC PLC Profiles, other honeypots for the same PLC models that were publicly exposed to the Internet and Gaspot, Conpot, S7commTrace, and SCADA HoneyNet. For such a purpose, we leveraged Shodan to gather data of Internet-facing *real* PLCs and PLCs flagged as honeypots. We looked at open ports, geolocation, Honeyscore, PLC model and IP addresses. Later, we compared these data to the one obtained for our HoneyPLC PLC Profiles. Once deployed to the Internet, it took about a week for Shodan to index our honeypots and identify the S7comm and HTTP services on ports 102 and 80.

5.4.3 Results

The results of our Shodan experiment, depicted in Fig. 8, show 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. Moreover, our S7-1200 and S7-1500 profiles got a 0.3 Honeyscore, which is comparable with the one obtained by *real* S7-1200 PLCs as indexed by Shodan. Unfortunately, at the time this experiment was performed, we were not able to find any S7-1200 honeypots in Shodan for comparison. Regarding the other four AWS instances, S7commTrace, Gaspot, and SCADA HoneyNet were not indexed by Shodan as they crashed when Shodan's crawler tried to interact with them. Thus, they could not be assigned a Honeyscore. Conpot, however, was successfully indexed and

was assigned a 0.3 Honeyscore. Overall, these results add compelling evidence with respect to Question Q-2, showing that HoneyPLC is effective at maintaining covertness against state-of-the-art reconnaissance tools.

5.5 Step7 Manager Experiment

We designed an experiment to test the capabilities of the HoneyPLC S7Comm Server, discussed in Sect. 4.3, against Step7 Manager [4], a Siemens proprietary software used to configure, write, and upload ladder logic programs to PLCs. For comparison purposes, we attempted to perform the same experiment on Conpot, the SCADA HoneyNet, and S7commTrace, which claim support for the S7comm protocol, as shown in Table 2.

5.5.1 Environment Description

For this experiment, we used a Windows XP virtual environment installed on a *desktop* host. Additionally, we installed HoneyPLC, the related work honeypots shown in Table 2, and all three Siemens PLC Profiles in different Ubuntu 18 LTS VMs and connected them to the Windows XP host.

5.5.2 Methodology

To test the compatibility of a particular honeypot with Step7 Manager, we performed the following: first, we attempted a direct, initial connection to the tool by using the 'Go Online' GUI feature. Second, we used Step7 Manager to list all the memory blocks contained within a given honeypot. Third, we also tried to upload a memory block to each honeypot, and finally, in a reciprocal action, we tried to download the contents of a memory block, which was previously stored by each honeypot under test.

5.5.3 Results

Our results show that HoneyPLC is the only implementation capable of handling all of the functionality previously mentioned, as is shown in Table 2. Conpot, S7commTrace, and SCADA HoneyNet were able to establish the initial connection, and however, the Step7 Manager threw a connection timeout error, preventing any further interaction and resulting in an aborted execution. Moreover, as S7commTrace is a high-interaction honeypot that implements features similar to the ones provided by HoneyPLC's S7comm Server, we strove to provide an extended comparison between them. The HoneyPLC S7comm Server improves over

Table 4 Comparison of S7comm function codes

S7comm implementation	Functions	Subfunctions
HoneyPLC	13	18
S7commTrace	12	14

S7commTrace by providing more functions and subfunctions as shown in Table 4. Specifically, it adds an error response function and insert block, delete block, blink LED, and cancel password subfunctions. The error response function and the delete and insert block functions, in particular, are important when injecting ladder logic programs and connecting with Step7 Manager. Overall, besides providing compatibility with Step7 Manager, HoneyPLC also provides enhanced capabilities for capturing ladder logic, e.g., reading and writing memory blocks, which are not supported by S7commTrace.

5.6 Internet Interaction Experiment

In order to explore the capabilities of HoneyPLC to interact with external, non-controlled agents, e.g., attackers, we designed an experiment intended to expose the PLC Profiles discussed in previous experiments to remote connections via Internet.

5.6.1 Environment Description

We leveraged the environmental setup we designed for our previous Shodan-based experiment in Sect. 5.4. Also, we used the same AWS EC2 instances equipped with PLC Profiles for the S7-300, S7-1200, and S7-1500 PLCs.

5.6.2 Methodology

We exposed the EC2 instances to the Internet for a period of 5 months. Using the HoneyPLC logging capabilities discussed in Sect. 4.5, we logged all received interactions. Later on, we analyzed such logs and obtained the results we discuss next.

5.6.3 Results

As a result of this experiment, more than 5GB of data were recorded. Table 5 shows the different S7comm function commands received by each PLC Profile. The fact that we recorded these functions means that external agents interacted with HoneyPLC beyond a simple connection performing reconnaissance tasks. Additionally, we received 4 PLC Stop functions on our S7-300 Profile, which stops

Table 5 S7comm function commands received

PLC profile	Setup communication	Read SZL	PLC stop	List blocks
S7-300	600	1013	4	80
S7-1200	202	324	0	0
S7-1500	292	343	0	0

Table 6 HTTP and SNMP interactions received

PLC profile	HTTP conversations	HTTP login attempts	SNMP get requests
S7-300	2060	205	1925
S7-1200	1791	30	567
S7-1500	13	0	1271

the current ladder logic program execution, suggesting that external agents tried to disrupt the PLCs' operation. Table 6 shows that our honeypots also received thousands of HTTP conversations and logged multiple HTTP authentication attempts on their administration websites, including the usernames and passwords used by the external parties. These authentication attempts could have been made by web crawlers or malicious actors trying different well-known or default passwords to log into the PLCs admin website. Additionally, we also recorded thousands of SNMP get requests that downloaded our PLC Profile's MIBs several times. Table 7 shows the distribution of S7comm connections based on geographical location. It can be noted that countries with most connections have historically been either the target or the initiators of attacks against ICS [14] recorded in the literature. Finally, at the time of writing this chapter, no attempts to inject malicious ladder logic into our honeypots were recorded. Such an attack would have been signaled by an attempt to write a memory block inside a PLC. Despite this limitation, the amount and nature of the interactions obtained provide additional support for affirmatively answering Question Q-2, showing that HoneyPLC can effectively engage external agents and tools.

5.7 Ladder Logic Capture Experiment

Finally, we were interested in exploring the capabilities of HoneyPLC to properly collect Ladder Logic malware that is injected by attackers, following the Case Scenario described in Sect. 4.1.

5.7.1 Environment Description

For this experiment, we leveraged the same HoneyPLC AWS test environment described in Sect. 5.4 for our Shodan experiment. Additionally, we locally deployed

Table 7 S7comm connections received by geolocation

Geo-location	S7-300	S7-1200	S7-1500	Geo-location	S7-300	S7-1200	S7-1500
United States	359	142	250	Netherlands	22	13	11
United Kingdom	2	1	3	Japan	8	2	2
Turkey	2	0	1	Italy	1	0	0
Switzerland	3	1	1	Iceland	1	1	2
Sweden	1	1	1	Hong Kong	2	1	1
South Korea	1	0	0	Germany	18	9	12
Slovakia	0	1	1	France	10	5	7
Singapore	4	3	5	Denmark	1	1	0
Russia	28	12	14	China	42	16	26
Romania	6	2	4	Canada	3	2	3
Poland	1	0	0	Bulgaria	2	1	0
Panama	2	1	3	Belize	3	3	3

Fig. 9 Ladder Logic payload example found in the Stuxnet malware

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

Conpot, Gaspot, S7commTrace, and SCADA HoneyNet. For Gaspot, we downloaded the latest version from GitHub [15] and installed it in an Ubuntu 18 LTS host. Next, for Conpot, we also downloaded the latest version from GitHub [30] and installed it from source in an Ubuntu 18 LTS host. Finally, we deployed the latest version of the SCADA HoneyNet [39] also in an Ubuntu 18 LTS. We faced some problems deploying the SCADA HoneyNet as it is currently not maintained at all (the latest version was released in 2004), and however, we were able to deploy the S7comm portion of the honeypot, enabling us to conduct this experiment. To test our implementation, we employed PLCinject [18], a research tool published by the SCADACS team, which is capable of injecting arbitrary compiled ladder logic programs into a PLC memory block. Figure 9 shows a sample of the ladder logic code dropped by the Stuxnet malware. We also set up a laptop host with Ubuntu 18.04 LTS installed with the latest version of PLCinject available on GitHub [41]. Since PLCinject also leverages the Snap7 framework, we installed a custom library and compiled PLCinject from source. We also used the Windows XP host described in Sect. 5.5 with Step7 Manager.

5.7.2 Methodology

Figure 10 illustrates our setup and methodology. The PLCinject host contains the ladder logic program sample that PLCinject will upload into HoneyPLC, which

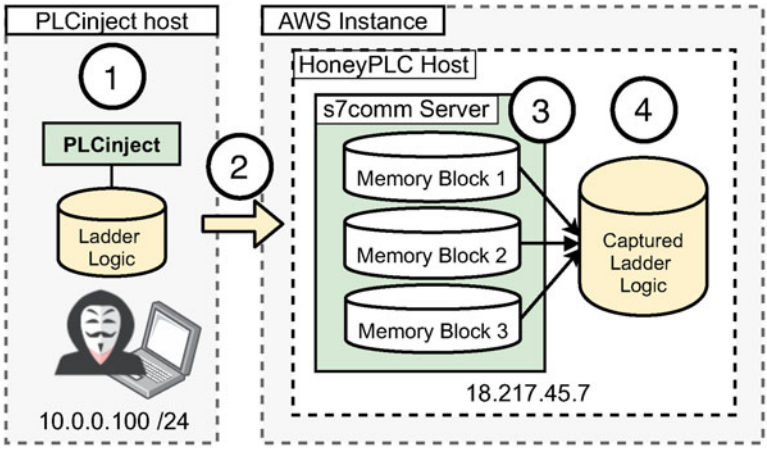

Fig. 10 Capturing Ladder Logic: initially, the attacker selects a malicious program and leverages PLCinject (1), which then establishes communication with an AWS instance running HoneyPLC (2). Malicious code is injected into a previously selected memory block exposed by the S7comm server (3) and finally written into a file repository (4)

resides inside an AWS instance exposing a set of standard PLC memory blocks. We leveraged the capabilities of PLCinject to connect and interact with the HoneyPLC host, eventually injecting the desired Ladder Logic program by using the command line. Later, using the Step7 Manager GUI, we created a new project and wrote a sample ladder logic to be injected into HoneyPLC. Next, we used the Step7 Manager to list the available memory blocks and then use the upload function to inject the sample ladder logic program into HoneyPLC. Later, we conducted another set of experiments focused on Gaspot, Conpot, S7commTrace, and the SCADA HoneyNet. We configured each of the honeypots with the correct IP addresses and ports and used PLCinject and the Step7 Manager to write the sample program into them, following the same process used for HoneyPLC.

5.7.3 Results

Our experiments were successful as we were able to inject a sample ladder logic program into HoneyPLC using both, PLCinject and the Step7 Manager. After the injection was completed, we logged into our honeypot file system and found the ladder logic file with its corresponding timestamp, which matched the contents of the blocks previously updated to PLCinject, as described in the previous paragraph. More to the point, after the Step7 Manager injection was completed, we downloaded our own sample program from HoneyPLC's S7comm server and used the ladder logic editor (included with Step7 Manager) to corroborate that our sample program was in fact saved in HoneyPLC's S7comm server. It is worth mentioning that the Step7 Manager did not crash or threw any errors while interacting with HoneyPLC's

S7comm server. This adds evidence as to the level of interaction that HoneyPLC provides. Regarding the Gaspot honeypot, our results show that it is not possible to inject any program into it. In fact, the TCP connection times out, and there is no reply. The results from Conpot show that it can, in fact, open a connection to TCP port 102, and however, it is reset, and the program upload cannot continue. S7commTrace results in the S7comm connection not being established. Finally, the S7comm portion of the SCADA HoneyNet accepts the TCP port 102 connection and starts the upload function needed to upload the ladder logic program, and however, after the upload function ends, there is no data saved or even transmitted. There results provide evidence for answering Question Q-3 affirmatively.

6 Discussion and Future Work

Before rounding up this chapter, we now present an extended discussion on the novelty, the features, and the experimental results obtained using HoneyPLC, as presented in previous sections. Also, we engage in a short discussion on the observed shortcomings of our approach and discuss interesting topics for future work that may benefit from using HoneyPLC as a supporting framework.

6.1 Comparing HoneyPLC with Previous Approaches

Following the comparison shown in Table 1, HoneyPLC provides significant improvements over the current state of the art of honeypots for PLCs. First, HoneyPLC provides better covertness capabilities than the ones provided by related works in the literature, as shown in the experimental procedures summarized in Table 2. Moreover, as detailed in Sect. 4.3, HoneyPLC provides advanced TCP/IP simulation based on Honeyd, plus the careful simulation of different domain-specific protocols. Whereas the simulation of various protocols is shared by many approaches in the literature, only HoneyPLC and SCADA HoneyNet [39] leverage the rich simulation features provided by the Honeyd framework. Second, the extensibility features of HoneyPLC, discussed in Sect. 4.2, allow for the effective simulation of different PLCs deployed in practice, as it was shown in the experimental procedures detailed in Sect. 5.2. Such a feature is not shared by any other approach in the literature, as shown in Table 1. Only a few approaches provide limited extensibility features, but those are mostly based on manually changing some configuration settings for the PLCs they support. As shown in Sect. 4.2, the HoneyPLC's Profiler Tool supports the collection and configuration settings for different *real* PLCs, which may allow for practitioners to create and distribute PLC Profiles for HoneyPLC for many different brands and models used in practice. Finally, HoneyPLC's Ladder Logic Capture feature is optimal for the understanding and analysis of malicious programs tailored for PLCs, which is not provided by any other related work, as shown in Table 2.

6.2 Limitations

Despite the innovative features of HoneyPLC and the promising evaluation results shown in Sect. 5, we identified the following limitations to our approach. First, as shown in Table 1, HoneyPLC does not provide support for modeling physical interactions as depicted by PLCs in practice. To solve this, future versions of HoneyPLC may be enhanced with a generic, general-purpose framework that facilitates the collection and subsequent modeling of physical interactions that can further engage and deceive attackers. Second, despite numerous attempts, we were unable to test HoneyPLC against Stuxnet, shown in Sect. 2.3, up to the point in which PLCs are injected with Ladder Logic code. This problem was also encountered by seasoned partners in industry, as it was revealed to us in private conversations. As an alternative, we strove to replicate a similar code injection scenario as shown in Sect. 5.7. Finally, as discussed in Sect. 5.6, we were not able to capture any Ladder Logic code injection attempts while exposing HoneyPLC to the internet during an extended period of time. We believe that such a thing may not necessarily represent a limitation in the capabilities of our approach, as shown in Sect. 5.7. However, we agree that future work focused on capturing instances of malicious code may obtain significant evidence of the suitability of HoneyPLC for engaging and deceiving external agents.

6.3 Future Work

First, we plan to add support to other ICS specific network protocols such as Modbus, which is widely implemented by other approaches in the literature. Second, we plan to expand the PLC Profile Repository of HoneyPLC, which is graphically depicted in Fig. 2 as an important part of our approach, to include several different PLC Profiles simulating other *real* PLCs widely used in practice, which may have been produced by different manufacturers and may include a diverse set of configuration options. We believe such a feature will likely increase the impact of HoneyPLC in many different projects in the research community, as well as in real-life ICS environments. Third, we plan to use HoneyPLC as a basis for simulating rich ICS infrastructures completely in software, modeling components like SCADA and other devices. Current ICSs are proprietary, closed, and composed of a plethora of costly devices, which clearly complicates the effective development and testing of new protection tools by researchers. In such regard, we believe that HoneyPLC can be combined with other emerging technologies such as *software-defined networks* (SDN) [22], to produce an automated, highly configurable, and automated approach effectively simulating ICS environments. Finally, we plan to turn HoneyPLC into a comprehensive suite for malware analysis for ICS by incorporating Ladder Logic analysis tools such as ICSREF [17], as well as other works such as PLCinject, featured in Sect. 5.7.

7 Conclusions

Attacks targeting ICS are now more real than ever and their consequences may be catastrophic. In such regard, honeypots help us understand and prepare for these attacks, and however, current implementations do not allow us to analyze and tackle brand new threats as desired. To overcome this situation, we have introduced HoneyPLC, a convenient and flexible honeypot, which significantly pushes the *state of the art* of the field forward. Additionally, we have provided experimental evidence that demonstrates that HoneyPLC outperforms existing honeypots in the literature, achieving a performance level comparable to *real* PLC devices. Finally, the HoneyPLC advanced extensibility features, which may allow HoneyPLC to better serve the heterogeneous world of ICS. As an example, we expect for practitioners to create and openly distribute many new PLC Profiles for a variety of PLCs used in practice, thus positioning HoneyPLC not only as a helpful tool for preventing and deterring ongoing attacks but also as the starting point for designing and evaluating new protection technologies for mission-critical cyber-physical systems and infrastructure.

Acknowledgments This work was supported in part by the National Science Foundation (NSF) under grant 1651661, the Department of Energy (DoE) under grant DE-OE0000780, the Army Research Office under grant W911NF-17-1-0370, the Defense Advanced Research Projects Agency (DARPA) under the agreements HR001118C0060 and FA875019C0003, the Institute for Information & communications Technology Promotion (IITP) under grant 2017-0-00168 funded by the Korea government (MSIT), a grant from the Center for Cybersecurity and Digital Forensics (CDF) at Arizona State University, and a grant from Texas A&M University—Corpus Christi. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or any agency thereof.

References

1. ABB: Plc automation. https://new.abb.com/plc. Accessed: 2020-02-24
2. Allen-Bradley: Programmable controllers. https://ab.rockwellautomation.com/Programmable-Controllers. Accessed: 2020-02-24
3. Antonioli, D., Agrawal, A., Tippenhauer, N.O.: Towards high-interaction virtual ics honeypots-in-a-box. In: Proc. of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy, pp. 13–22 (2016)
4. Berger, H.: Automating with STEP7 in STL and SCL: programmable controllers Simatic S7-300/400. Publicis (2006)
5. Buza, D.I., Juhász, F., Miru, G., Félegyházi, M., Holczer, T.: Cryplh: Protecting smart energy systems from targeted attacks with a plc honeypot. In: Int. Workshop on Smart Grid Security, pp. 181–192. Springer (2014)
6. Cao, J., Li, W., Li, J., Li, B.: Dipot: A distributed industrial honeypot system. In: Int. Conference on Smart Computing and Communication, pp. 300–309. Springer (2017)
7. Case, D.U.: Analysis of the cyber attack on the Ukrainian power grid. Electricity Information Sharing and Analysis Center (E-ISAC) vol. 388 (2016)
8. Cybersecurity (CISA) I.S.A.: Apt cyber tools targeting ics/scada devices (2022). https://www.cisa.gov/uscert/ncas/alerts/aa22-103a

9. DataSoft/Honeyd: (2020). https://github.com/DataSoft/Honeyd. Original-date: 2011-12-09T22:40:03Z
10. Dragos, I.: Crashoverride: Analysis of the threat to electric grid operations (2017). Online: https://dragos.com/blog/crashoverride/CrashOverride-01.pdf
11. Dragos, I.: Chernovite's pipedream malware targeting industrial control systems (ics) (2022). https://www.dragos.com/blog/industry-news/chernovite-pipedream-malware-targeting-industrial-control-systems/
12. Falliere, N., Murchu, L.O., Chien, E.: W32. stuxnet dossier. White paper, Symantec Corp., Security Response **5**(6), 29 (2011)
13. Hahn, A.: Operational technology and information technology in industrial control systems. In: Cyber-security of SCADA and Other Industrial Control Systems, pp. 51–68. Springer (2016)
14. Hemsley, K.E., Fisher, E., et al.: History of industrial control system cyber incidents. Tech. rep., Idaho National Lab.(INL), Idaho Falls, ID (United States) (2018)
15. Hilt, S.: Gaspot released at blackhat 2015 (2016). https://github.com/sjhilt/GasPot
16. Jicha, A., Patton, M., Chen, H.: Scada honeypots: An in-depth analysis of conpot. In: 2016 IEEE Conference on Intelligence and Security Informatics (ISI), pp. 196–198. IEEE (2016)
17. Keliris, A., Maniatakos, M.: ICSREF: A framework for automated reverse engineering of industrial control systems binaries. In: Network and Distributed System Security Symposium, (NDSS). The Internet Society (2019)
18. Klick, J., Lau, S., Marzin, D., Malchow, J.O., Roth, V.: Internet-facing plcs-a new back orifice. Blackhat USA, pp. 22–26 (2015)
19. Kneschke, J.: Lighttpd-fly light. https://www.lighttpd.net/ (2020)
20. Kołtyś, K., Gajewski, R.: SHaPe: A honeypot for electric power substation. J. Telecomm. Inf. Technol. (4), 37–43 (2015)
21. Langner, R.: Stuxnet: Dissecting a cyberwarfare weapon. IEEE Secur. Priv. **9**(3), 49–51 (2011)
22. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: Rapid prototyping for software-defined networks. In: Proc. of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX. Association for Computing Machinery, New York, NY, USA (2010)
23. Lian, F.L., Moyne, J., Tilbury, D.: Network design consideration for distributed control systems. IEEE Trans. Control Syst. Technol. **10**(2), 297–307 (2002)
24. Litchfield, S., Formby, D., Rogers, J., Meliopoulos, S., Beyah, R.: Rethinking the honeypot for cyber-physical systems. IEEE Internet Comput. **20**(5), 9–17 (2016)
25. López-Morales, E., Rubio-Medrano, C., Doupé, A., Shoshitaishvili, Y., Wang, R., Bao, T., Ahn, G.J.: HoneyPLC: A next-generation honeypot for industrial control systems. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS '20, pp. 279–291. Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3372297.3423356
26. Lyon, G.F.: Nmap network scanning: The official Nmap project guide to network discovery and security scanning. Insecure (2009)
27. Matherly, J.: Complete guide to shodan. Shodan, LLC (2016-02-25), vol. 1 (2015)
28. Matherly, J.: Personal communication (2019)
29. Mokube, I., Adams, M.: Honeypots: concepts, approaches, and challenges. In: Proc. of the 45th Annual Southeast Regional Conference, pp. 321–326 (2007)
30. MushMush: Conpot (2020). https://github.com/mushorg/conpot
31. Nardella, D.: Snap7 (2018). http://snap7.sourceforge.net/
32. nmap.org: Os detection (2022). https://nmap.org/book/man-os-detection.html
33. Provos, N.: Honeyd-a virtual honeypot daemon. In: 10th DFN-CERT Workshop, Hamburg, Germany, vol. 2, p. 4 (2003)
34. Provos, N., Holz, T.: Virtual Honeypots: From Botnet Tracking to Intrusion Detection. Pearson Education (2007)
35. Repository, U.M.: snmpwalk - retrieve a subtree of management values using snmp getnext requests (2019). http://manpages.ubuntu.com/manpages/bionic/man1/snmpwalk.1.html
36. Repository, U.M.: Wget - the non-interactive network downloader (2019). http://manpages.ubuntu.com/manpages/disco/en/man1/wget.1.html

37. Response, S.I.: Dragonfly: Cyberespionage attacks against energy suppliers. Tech. Rep., July (2014)
38. S7comm - The Wireshark Wiki (2016). https://wiki.wireshark.org/S7comm
39. SCADA HoneyNet Project: Building Honeypots for Industrial Networks (2020). http://scadahoneynet.sourceforge.net/
40. SCADACS: Snap7 (2017). https://github.com/SCADACS/snap7
41. SCADACS/PLCinject (2020). https://github.com/SCADACS/PLCinject. Original-date: 2015-07-13T09:38:19Z
42. Schwartz, M.D., Mulder, J., Trent, J., Atkins, W.D.: Control system devices: Architectures and supply channels overview. Sandia Report SAND2010-5183, Sandia National Laboratories, Albuquerque, New Mexico, vol. 102, 103 (2010)
43. Searle, J.: plcscan (2015). https://github.com/meeas/plcscan
44. Shi, J., Wan, J., Yan, H., Suo, H.: A survey of cyber-physical systems. In: 2011 International Conference on Wireless Communications and Signal Processing (WCSP), pp. 1–6 (2011). https://doi.org/10.1109/WCSP.2011.6096958
45. 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
46. Slowik, J.: Anatomy of an attack: Detecting and defeating crashoverride. VB2018, October (2018)
47. Stouffer, K., Falco, J., Scarfone, K.: Nist special publication 800-82: Guide to industrial control systems (ics) security. National Institute of Standards and Technology (NIST), Gaithersburg, MD (2008)
48. Stouffer, K., Lightman, S., Pillitteri, V., Abrams, M., Hahn, A.: Nist special publication 800-82, revision 2: Guide to industrial control systems (ics) security. National Institute of Standards and Technology (2014)
49. Wade, S.M.: Scada honeynets: The attractiveness of honeypots as critical infrastructure security tools for the detection and analysis of advanced threats (2011)
50. Wilhoit, K., Hilt, S.: The gaspot experiment : Unexamined perils in using gas-tank-monitoring systems. GitHub repository (2020)
51. Xiao, F., Chen, E., Xu, Q.: S7commtrace: A high interactive honeypot for industrial control system based on s7 protocol. In: Int. Conference on Information and Communications Security, pp. 412–423. Springer (2017)