On Identifying Repeated Patterns of OT Attacks with LOGistICS

Stefano Bistarelli[1,†], Emanuele Bosimini[2] and Francesco Santini[1,*,†]

[1] Department of Mathematics and Computer Science, University of Perugia, Perugia, Italy
[2] Aruba S.p.A., Ponte San Pietro (Bergamo), Italy

Abstract

LOGistICS is a novel monitoring framework for investigating the security of industrial PLC systems. Diverse processing components and probes with different tasks are included in the architecture. Reconstructing the process of cyber-attacks against industrial devices in its entirety, and also identifying the actors, is the topic of this study. Our solution is suitable for research and production environments, allowing for more interaction with an attacker while remaining undetectable as we will see from the behaviour of the actors. To achieve this aim, we exposed LOGistICS to the Internet with a module that we developed to examine suspicious activities, sequencing the operations carried out by the same attacker over time and therefore identifying its pattern. We found that the Cyber Kill Chain model is not always respected. We discuss these anomalies and provide our insights.

Keywords

Operational Technology, Honeypot, Attack behavior

1. Introduction

Operational Technology (OT) is a term that collectively refers to hardware and software that synergically cooperate to pursuit an industrial goal. Initially, *Industrial Control Systems* (ICS) [1] were isolated by nature, being limited to the process network. Security was ensured by both obscurity and isolation. The protocols were proprietary and related documentation was not publicly available. We could forcefully say that the systems, not being connected to the internet, were safer. However, nowadays, in the era of Industry 4.0, ICS are able to be interconnected in a production intranet, and consequently exposed to the Internet through the company network. This need for connection, also combining the questionable paradigm of making open standards, clearly exposes ICS to serious safety problems. Due to the escalation of the Russian-Ukrainian conflict, it is logical to assume a substantial increase of cyber-attacks on critical infrastructures. In fact, Russian attackers targeted energy plants in Ukraine with the aim of triggering a power grid failure.[1,2] The mal-

ITASEC'22: Italian Conference on Cybersecurity, June 20–23, 2022, Rome, Italy

*Corresponding author.

† The authors are members of the INdAM Research group GNCS and of Consorzio CINI.

stefano.bistarelli@unipg.it (S. Bistarelli); emanuele.bosimini@staff.aruba.it (E. Bosimini); francesco.santini@unipg.it (F. Santini)
0000-0001-7411-9678 (S. Bistarelli); 0000-0002-3935-4696 (F. Santini)

[1] "Ukraine energy grid hit by Russian Indestroyer2 malware": http://bitly.ws/rXIN.

[2] "Industroyer2, why the Russian attack on the Ukrainian electricity grid is so important": http://bitly.ws/rXJ7.

ware used for this purpose was *Industroyer2*, which targets the controller hardware that manages the flow of water, use of cleaning agents and other embedded machines that keep water systems running efficiently [2]. We can think about resorting to protection measures adopted in conventional ICT systems, such as a Firewall or an Intrusion Detection System. However, this introduces design-related safety issues. The priority of an ICT system is to focus on confidentiality and data integrity, but, on the other hand, ICS are designed to ensure *reliability/availability*, even jeopardizing the *confidentiality* and *integrity* of data. Further common goals in ICS are related to the *safety* of operators and environment, and, of course, *productivity*. An example of such different security concerns with respect to ICT systems is represented by the Modbus protocol [3], a long-lived and reliable protocol, which nonetheless comes without authentication mechanisms. Nowadays the devices and protocols adopted in an ICS are used in nearly every industrial sector and critical infrastructure such as the manufacturing, transportation, energy, and water treatment industries. Given all the introduced peculiarities and the importance of monitoring security in such applications, in this paper we present the enhanced implementation of LOGistICS an emulation and monitoring system for OT [4], and the experimental analysis conducted through the behavioral module. In particular, we focus on the benefits of using our solution to identify the attitude of the actors attacking industrial protocols, such as S7Comm from Siemens [5] and Modbus from Schneider Electric [3], which are also used extensively in home and building automation. The framework we propose is composed of *i)* a highly customizable *honeypot* that emulates the two aforementioned ICS services, *ii)* a sniffer that records traffic, and *iii)* a monitoring node, which is capable of analyzing and plotting recorded data. A fourth component implements malicious activities towards the honeypots, being an assessment node not properly part of the monitoring platform (more a test node). What is described in the following goes under the umbrella of *deception technology*, which is an emerging category of cybersecurity defense. Deception technology automates the creation of traps (decoys), and can detect, analyze, and defend against zero-day and advanced attacks, often in real time.

Honeypots [6] are physical or virtual systems used as "baits" to decoy the attackers. Their goal is to expose only the vulnerable interface of a full service (e.g. SSH), without implementing all of its logic and features. Honeypots can be used to attract and record attacks, allowing the underlying patterns to be identified [7]. The interaction metrics are widely used to characterize honeypots. Low-interaction honeypots give limited responses: they're mostly employed for statistical analysis, and they're good enough to spot spikes in the number of requests, such as those caused by autonomous malware. In comparison to low-interaction honeypots, medium interaction honeypots give a higher level of interaction for the attacker: their purpose is to generate appropriate responses in the hopes of initiating follow-up attacks. High-interaction honeypots, on the other hand, expose more resources than previous ones and capture the most amount of data imaginable, including comprehensive attack logs. LOGistICS is light, stealthy, isolated through containerization technologies, platform independent, easy to deploy, with a fair degree of interaction. Our framework leverages two libraries that allow communication via S7Comm and Modbus protocols: *Snap7*[3] and *Pymodbus*[4] respectively. With the purpose to deceive search engines, we made some low-level changes to make the behavior of such honeypots more faithful to a real Programming Logic Controller (PLC). We prove its camouflage ability with reconnaissance

[3]Snap7: snap7.sourceforge.net.
[4]Pymodbus: pymodbus.readthedocs.io/en/latest.

Honeypot	Modbus/S7Comm	Behavioral Detection	Interaction Level	Cross-platform	Extensibility	No. supported PLCs
CryPLH	✗/✓	✗	High	n.a.	✗	1
HoneyPLC	✗/✓	✗	Medium	✓	✓	5
Gaspot	✗/✗	✗	Low	✗	✗	1
HoneyVP	✗/✓	✗	High	✓	✓	1
S7CommTrace	✗/✓	✗	High	n.a.	✓	1
Conpot	✓/✓	✗	Low	✓	xml	1
LOGistICS	✓/✓	✓	Medium	✓	csv	7

Table 1
A comparison with existing ICS honeypots.

and honeypot detection tools. Detection systems likely use unique characteristics of specific honeypots in order to identify them, such as the property-value pairs of default honeypot configurations. The majority of the honeypots discussed in Section 2 have a low level of interactivity. Quantitative traffic analysis can benefit from this level of involvement. However, to analyze the state of the art of attacks we cannot rely on this approach. According to Shodan,[5] the most common mistake of researchers deploying honeypots is to use a default configuration without applying changes [8]. All default configurations return the same banner, including same PLC names, same serial numbers etc. Around 30% of Siemens S7 PLCs connected to the internet have the same serial number, which corresponds to the default serial number of a Conpot instance.[6] Moreover, in the literature the only ICS honeypot that can be extended in a user-friendly way is *Conpot*,[7] via XML.

1.1. Paper structure

Some initial results have been presented in [4]; this paper extends them by proposing a refined architecture (see Figure 1) and describing the behavioral pattern of attackers. The remainder of the paper is organized as follows: Section 2 summarizes the related work in the literature and gives a minimal background to the reader. Section 3 reports necessary background notions about the two protocols at the core of LOGistICS: Modbus and S7Comm. Section 4 describes the implementation of the systems, while Section 5 reports some tests we performed to evaluate LOGistICS. Section 6 presents the behavior of attackers, that is the sequence of commands they executed on the medium-interaction honeypots. Finally, in Section 7 we conclude the paper with final thoughts and several ideas about possible future work.

2. Related Work

Gaspot is a project conducted in the Trend Micro research center,[8] and was later presented at the Blackhat 2015 conference [9].[9] This low interaction honeypot is designed to emulate the behavior of the Guardian AST, a remote monitoring system for gas tank. It is written in Python and some options can be modified, such as the tank name and volume. CryPLH simulates a Siemens Simatic S7-300 PLC. It emulates HTTP, HTTPS, S7Comm and SNMP services. Furthermore, the TCP/IP

[5]Shodan: shodan.io.
[6]The complete guide to Shodan: ia800705.us.archive.org/17/items/shodan-book-extras/shodan/shodan.pdf.
[7]Conpot: conpot.org.
[8]Trend Micro: https://www.trendmicro.com.
[9]Black Hat: https://blackhat.com.

Stack is simulated via the Linux kernel [10]. The authors identify CryPLH as a high-interaction honeypot. As suggested in [11], the attacker can never fully interact with the emulated PLC. The authors set the maximum level of security by rejecting any authentication attempt. As a result, [11] and [12] downgrade it as a low interaction honeypot. HoneyVP is a HoneyPLC-inspired hybrid honeypot that redirects inbound industrial traffic from the cloud to real Siemens PLCs [13]. Conpot is an open-source low-interaction honeypot that emulates a wide range of OT and IT protocols. It emulates a Siemens S7-200 PLC by default [14], but it also includes some other profiles that can be customized. Morales et al. developed HoneyPLC, an ICS honeypot based on Honeyd [15]. It emulates three protocols supported by Siemens PLCs: S7Comm, SNMP and HTTP. HoneyPLC provides multiple PLCs profiles and it implements the capture of *Ladder Logic* programs. S7CommTrace is a honeypot developed by Feng Xiao et al. that emulates S7-300 PLC [5]. It is extensible with a user template that defines the PLC fingerprint. Our solution, in addition to providing a medium interaction honeypot capable of emulating two widely used ICS services, is cross-platform, and provides a user-friendly graphical interface for PLC deployment. Moreover, LOGistICS also supports 7 types of PLC out-of-the box. Based on the features and the limitations of the previous contributions, in Table 1 we report a comparison of ICS honeypots.

3. Modbus and S7Comm

In this section we introduce the necessary background notions about Modbus and S7Comm, which are the two protocols monitored by LOGistICS. We mostly focus on the functions that can be invoked on a PLC, which will be exploited during attacks (see Section 5 and Section 6).

3.1. Modbus

Modbus is a protocol developed by Modicon [16] in 1979. The creator company was subsequently acquired by Schneider Electric. In 2004 the rights on the Modbus protocol were transferred from Schneider Electric to the Modbus organization. This transition made it possible to make Modbus an open protocol and to consequently adopt it on a large scale of devices (from intelligent sensors to PLCs) and to ride the wave of the IoT [17]. Modbus is implemented with a master/slave architecture, where the entities can perform simultaneous requests. We can divide Modbus messages into four categories [18]. *Request* message is how the master establishes the transmission, sending the request message to the slave. *Response* message is how the slave responds back to the master with the requested data. Upon master receives a response, it sends a *confirmation* message to the queried slave. Then an *indication* message is generated by the slave and confirms the receipt of master request. Modbus operates at the application level, and for this reason it is interoperable with the underlying levels by adapting the *Application Data Unit* (*ADU*) structure. The Modbus *PDU* consists of the data field and function code. The ADU is made up of the PDU, in addition to the *Address and Error Check field.* In order to transport the PDU over TCP/IP (port 502) a header is added to identify the ADU: the *Modbus Application* (*MBAP*) header [19]. This 7-Byte header allows the packet identification in case of fragmentation. Modbus TCP [3] does not include the CRC expected in the ADU as the integrity is controlled by the higher levels. The data and function code fields are part of PDU, and they belong to every possible variant of architecture. The data field contains all the information useful for the slave in order to execute a request, and to respond adequately to the master.

Name	Code	Description
Read coils	1	Requests the ON/OFF status of discrete coils
Read Discrete Inputs	2	Requests the ON/OFF status of discrete inputs
Read Holding Registers	3	Retrieves the contents of the holding registers
Read Input Registers	4	Retrieves the contents of input registers
Write Single Coil	5	Changes the ON/OFF status of a single coil
Write Single Holding Register	6	Changes the content of a single holding register
Force Multiple Coils	15	Writes the contents on a range of discrete coils
Force Multiple Holding Register	16	Writes the contents on a range of holding registers

Table 2
Modbus function codes with their effect.

According to Schneider Electric documentation, the data field has a variable length depending on the type of function invoked. Function code represents the core meaning of the message, whose encoding denotes the action performed. Function codes are of three types: *public*, *user-defined* and *reserved*. Public function codes are documented and tested. This case also includes codes that are reserved for future developments. User-defined function codes are those codes not supported by the protocol specification, and which the user can implement from scratch. Reserved function codes are implemented by companies and are not available for public use. Modbus protocol provides one byte of code-mapped field capable of representing 255 possible functions: value "0" is not valid, values from 1 to 127 are effectively used, and values between 128 and 255 are used for exception function code in response messages. Table 2 shows a list of Modbus function codes.

3.2. S7Comm

S7Comm is a proprietary protocol developed by Siemens. In the industry it is used for diagnostic and PLC programming purposes. S7Comm works via TCP on port 102 and relies on the *Connection-Oriented Transport* (*COTP*) protocol and the *Transport Packet* (*TPKT*) service [20]. The packet is encapsulated with the COTP header in order to be ISO-on-TCP compliant (RFC1006) [21].

The S7 PDU consists of three core components: *header*, *parameters* and *data*, where the latter component is optional. S7Comm is a command oriented protocol, in the sense that each transmission contains a command or a response to it. Protocol documentation is not publicly available. For this reason projects such as the open-source communication library Snap7 [22] and Libnodave [23] have come to light. It is possible to transfer user programs or individual portions to a PLC. To interface data with programs like Siemens Step7,[10] PLCs use distinct parts, called blocks [24]. As introduced before, S7Comm is the payload of a COTP packet, and it is distinguished by the identifier 0x32 (as known as *Magic Byte*). Then we find *Message Type*, *Data Unit*, *Reserved, Parameter and Data length*, and finally error classes and codes. The Message Type, often identified as *ROSCTR*, can be interpreted in four ways. Job request is the request sent by the master. Ack signal is the acknowledgment sent by the slave, omitting the data field. Ack data is the acknowledgment sent by the slave with the optional data field, containing the Job. User data is the extension of the original protocol containing ID requests/responses. In the case of parameters like Job and Ack data, the optional parameter that follows is the function code. The remaining fields may vary depending

[10]Step7: cache.industry.siemens.com/dl/files/056/18652056/att_70831/v1/S7prv54_i.pdf.

on this parameter, which decides the purpose of the communication. The *Setup Communication* function is used to establish the first S7Comm connection, and provides information on Ack queues (i.e. number of parallel jobs without Ack) and the maximum length of the PDU. The *Read* and *Write* functions are self-explanatory, and are used to read and write more variables on the PLC. Functions such as *Request Download*, *Download Block*, *Download End*, *Start Upload*, *Upload* and *Upload End* are used to perform block download and upload operations. The *PLC Stop* function stops the execution of the program running on the PLC, and *Program Invocation* (PI) manages the execution of such a program in a wider way (it also include the PLC Stop functionality). The *CPU service* is used to access fields such as the protection levels of the CPU and the SZL partial lists.

4. Implementation

In this section we describe the components of LOGistICS framework. In Section 4.1 we describe the overall experimental setup in its entirety. In Section 4.2 we describe the sniffer we implemented. We mainly focus on its versatility, and on its ability to listen only to the traffic of interest. In Section 4.3 we describe the implementation of the Modbus service based on Pymodbus, by also explaining how we enriched the functionality of the emulated service. Compared to the honeypots seen in Section 2, we guarantee the user the ability to customize the PLC template while still ensuring considerable camouflage capabilities [4]. In Section 4.4 we describe the implementation of Snap7 based on S7Comm service. We discuss the client-side implementation that we managed in order to test locally the solution. We present a novel tool that deploys our implementation of the PLC in a simple way. Finally, in Section 4.5 we describe the implementation of the behavioral module which is able to reconstruct the command sequence of an attacker.

4.1. Architecture

The framework, whose architecture is represented in Figure 1, can be summarized by four main components logically and physically separated: honeypot node, sniffing node, assessment node and monitoring node [4].

The **assessment** node is equipped with Kali Linux. The purpose of this node is to evaluate in depth the interaction with the honeypot node with respect to the state of the art of ICS Red Team tools available in the Kali Linux distribution. The **honeypot** node exposes Modbus and S7Comm services. In order to isolate and to ensure portability, we distribute these services via container technology. For clarity, the only node actually exposed is the honeypot as we can see in the figure, and the ICS activities that we report in Section 5 and in Section 6 are external and real. The **sniffing** node is on the same site of the honeypot, on which we launched an instance of our sniffer. This sniffer perpetually and exclusively listens to the exposed services, and generates files in *pcap* format. We use an intrusion detection system that we have passively configured in such a way as to generate, starting from the captured packets, industrial events in human readable format [4]. Finally, the **monitoring** node stores the events obtained from the sniffing node. Monitoring node contains an *ELK stack*[11] instance. ELK is a group of open source products designed to help users take data from any type of source and in any format and search, analyze, and visualize that data (also in real time).

[11]The Elastic Stack: https://www.elastic.co/what-is/elk-stack.

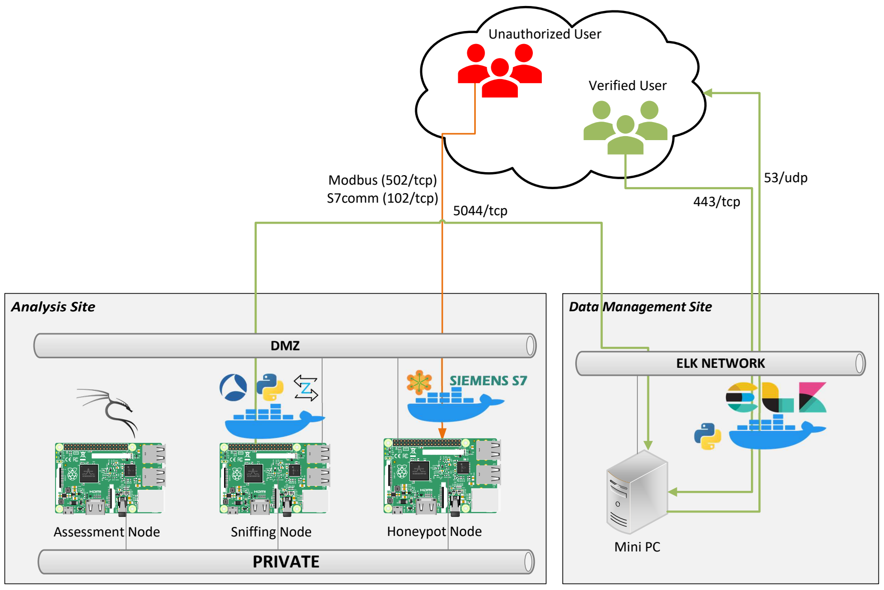


Figure 1: The architecture of LOGistICS (enhanced and redrawn from [4]).

In this paper we will not detail this node for the sake of space. The honeypot and monitoring nodes implement *Docker*[12] as the underlying container technology. We utilize Raspberry Pi hardware as a cross-platform proof of concept because honeypots and monitoring nodes need to be active for long periods of time, so low-power devices are preferable. Therefore, hardware infrastructure of LOGistICS is represented by Raspberry racks and mini PC designed to reduce energy consumption. Furthermore, the cost of OT devices is significantly higher than that of IT devices.

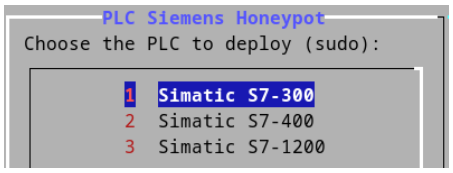
4.2. Packet capture

From the design point of view, since the analysis takes time, log optimization is a must-have. In order to meet this requirement, we created a sniffer able to analyze industrial traffic. Regarding packet creation and processing part, we used Pyshark. This allows to have a certain flexibility in selecting in a customizable manner which protocols should be filtered using BPF technology. By default, the sniffer captures packets from Modbus and S7Comm protocols on the loop-back interface. The protocol and the interface can be customized via Command-Line Interface (CLI).

4.3. Modbus emulation

It emulates the behavior of a PLC, which manages the filling and emptying of tanks through a system of electric pumps. With the purpose to make the honeypot more attractive for the attackers, we enriched the fingerprint and made it customizable [4]. Finally, we implemented four kinds of Modbus templates, which correspond to the identity of real PLCs.

[12]Docker: docker.com.



```
PLC Siemens Honeypot
Choose the PLC to deploy (sudo):

     1  Simatic S7-300
     2  Simatic S7-400
     3  Simatic S7-1200
```

Figure 2: LOGistICS Siemens S7 PLC deploy interface.

4.4. S7Comm emulation

LOGistICS supports three types of Siemens PLCs: *S7-300*, *S7-400* and *S7-1200*. To allow a user-friendly selection of the honeypot, we implemented a GUI as shown in Figure 2 capable of containerizing the selected class of PLC to be emulated, and quickly start it.

In the aforementioned Siemens PLCs, by default data-blocks are not protected and can accessed or written by users [25]. To test the degree of interaction before the exposure phase to the Internet (see Section 5), we implemented a C++ master that executed requests to the considered PLCs. Snap7 provides a sample master that is able to query a physical PLC: for example, by retrieving the PLC version and list of data-blocks. We enhanced the master with the purpose to enable the interaction with the password-protection mode of a PLC. We implemented a module in our Snap7 master, which returns three values concerning the CPU protection level, defined as Byte in the firmware. The array of Byte *SZL W_16_0232 index W_16_0004* contains the protection levels and position of the operating mode selectors. By exchanging messages between our own honeypot and the master, we were able to evaluate the security of the PLC, and thus modify the honeypot answer in order to make it authentic, i.e., more similar to a real PLC.

4.5. Behavioral Module

This module is implemented on the sniffing node, reconstruct a sequence of industrial commands invoked by the attacker in chronological order, whose is identified through IP address and autonomous system number (ASN) fields. The researcher is able to choose his time window, reducing or increasing depending on the type of analysis to conduct. In addition to attack patterns, the module can be well suited for analyzing sequence similarities [26].

5. Experiments

In order to evaluate LOGistICS, we deployed it towards the Internet for almost two months. The most frequent Modbus event we recorded was *Read Device Identification* invoked using function code 43. This encapsulation mechanism allows to transfer predefined and reserved fields: reserved function codes are implemented by companies and are not available for public use. In addition to being a field that is commonly read by search engines, it also corresponds to the first stage (i.e., reconnaissance) of the *ICS killchain* [27]. This stage is usually conducted both actively using reconnaissance tools, and passively through search engines such as Shodan or other services that allow for investigating the targeted resource without interacting directly with it. We were able to detect the stealth scans exception. In fact, using Wireshark we identified this kind of activity by following the

SYN$\rightarrow$ *SYN/ACK*$\rightarrow$ *RST* pattern [4]. This type of request was always performed by scanners and we believe it is therefore a benign request. We logged unauthorized reading requests to the *holding registers*, which store 16 bits readable and writable configuration-values. There are also exceptions related to the reading of input registers (i.e., 16 bits values representing measurements and statuses). We believe that a host has tried to access an undefined input, i.e. the register does not exist. Similarly, the same holds true for *Write Multiple Registers* exceptions. We detected many requests for writing to multiple registers, where unauthenticated hosts attempted to corrupt the honeypot, or were looking for some undefined behavior of our configuration. Regarding S7Comm events, we collected a large number of *Setup Communication requests*, used to establish a communication with the S7Comm Layer. Being able to query different fields in *Read SZL* requests, the attacker could be then encouraged to execute other following function codes: a medium-interaction honeypot inactivates possible attackers. In fact we were able to capture a *PLC Stop (0x29)* request and a *Write (0x05)* variable request on Data Block 1, which respectively stops the CPU and creates a boolean variable.

Remark 1. *We found that the peak of activities recorded was mostly due to TCP SYN scans. This was discovered thanks to the separation of logs by protocol type: in that one related to the application layer there was no trace of the performed-request type. However, by crossing these data with the ones related to the activities recorded in the presentation layer (ISO-COTP), we realized a high number of half-open connections*[13] *were recorded by the same actor having unknown DNS assigned to IP Volume inc. (ASN 202425). This host carried out this activity without actively obtaining information about our instance. We believe that the host already knew our configuration thanks to search engines like Censys.*

During the classification of industrial traffic, we found that there were many more potentially harmful Modbus events (274) than S7Comm (6), excluding TCP SYN scans as described in Remark 1. By referring to the Modbus service, we categorized the requests such as “Report Slave ID”, “Device ID” and “Half-Open Scan” as benign. However, we consider read requests other than those listed above to be disruptive, such as read coils and registers, which may contain confidential information. Concerning S7comm, we considered read requests to the S7Comm protocol as not malicious, since they are mostly performed by periodic scanners, with the exception of those that concern the CPU protection level. Any S7Comm write request, including exceptions, was clearly treated as malicious, regardless of the service. The high number of Modbus requests classified as malicious can be explained by the fact that *i)* the interaction with a Modbus PLC, by design, can involve multiple coils and registers, and *ii)* Modbus is now an open standard (unlike S7Comm), which makes it easier to write a script that brute-forces against all the registers.

6. Behavior of Attackers

In order to characterize the actors we outline its sequence of commands (i.e., the host behavior) grouped by IP address. For each host, the sequence of commands executed was captured only once, and there were no recurring activities from the same IP during the high-intensity periods. We discovered this by grouping events initially by a certain time slot, and subsequently by

[13]The lack of synchronization could be due to malicious intent, such as the TCP SYN Flood attack.

	IP	ASN	Organization	Pattern
P1	27.122.12.245	55536	Pacswitch	$2, 0, 1^{84}, 4, 5^{33}, 1^{82}$
P2	185.104.184.99	9009	M247	$2, 0, 6^{160}$
P3	93.174.95.106	29073	IP Volume	$3, 0, 6^{3}, 3, (0, 3)^{38}$
P4	71.6.199.23	10439	CARI.net	$3, 0, 6^{3}, (3, 0)^{14}$
P5	71.6.146.185	10439	CARI.net	$3, 0, 6^{3}, (3, 0)^{15}$

Table 3
Host sequence of commands related to Modbus service. The daily periods analyzed are: P1) January 6, P2) January 15, P3) January 25, P4) January 30, P5) February 5. The events are mapped as follows : 0) Read Device Identification, 1) Read Holding Registers, 2) Read Input Registers Exception, 3) Report Slave ID, 4) Write Multiple Registers, 5) Write Multiple Registers Exception, 6) Half-open scan.

day, noting that the sequence of commands remained unchanged. We believe this is due to the duration of the experiment, as in general the port scanners repeat the same activities on a monthly basis. We represented the encoded sequence of commands for hosts related to Modbus requests. We consider 5 of these command patterns as relevant, and we show them in Table 3, where the superscript on each command represents how many times it has been invoked. Each of these patterns (P) was logged on a different day, as reported by Table 3.

We can clearly distinguish the recurring scanners with the harassing actors during the initial phase of communication. The actors belonging to the CARI.net and IP Volume organization have an almost identical behavior, and start their activity by making Report Slave ID and Read Device Identification requests. We could remove the first two commands, as typically a couple of read requests (slave and device information) are needed to know how to compromise the device. But as we can see in Table 3, this can be argued. In fact, M247 and Pacswitch hosts execute Read Device Identification request only after making an illegal reading of the registers (note that M247 performed 160 TCP SYN requests, which may indicate a suspicious behavior). For what concerns the activities towards the S7Comm protocol, we identified in a similar way the most interesting patterns during the periods of greatest intensity. Differently to what emerged with Modbus activities, very similar S7Comm patterns were recorded belonging to hosts of the same organization. Moreover, some actors performed the exact same sequence during the period of intense activity. In order to simplify the representation, we assume with regard to the S7Comm activities, that the hosts of an Autonomous System have the same behavior.

In Table 4 we represent the sequence of commands performed by Organization hosts, grouped by high-intensity activity period. Regarding the S7Comm service, we were able to reconstruct

	IP	ASN	Organization	Pattern
P1	167.248.133.37 89.248.174.3	398722 202425	Censys, IP Volume	8, 2, 4, 5, 4, 5
P2	27.122.12.245	55536	Pacswitch	8, 2, 9, 3
P3	192.241.222.191	14061	DigitalOcean	8, 2, 4, 5, 4, 5

Table 4
Host sequence of commands related to S7Comm service. The daily periods analyzed are: P1) January 14 and January 24, P2) January 24, P3) January 30. The events are mapped as follows: 2) Setup Communication response, 3) Write variable response, 4) Read SZL request, 5) Read SZL response, 8) Setup Communication request, 9) Write variable request.

the complete communication between the actors and the honeypot, reporting in chronological order the response to each request made. As the results show, DigitalOcean hosts that make two Read SZL requests followed by the Setup Communication request are recurring scanners. The behavior of the Pacswitch host differs from the aforementioned host, which instead of running a port-scan tries to write directly to our honeypot. The unrecognized pattern of the host belonging to IP Volume inc. is explained in Remark 1.

7. Conclusion and Future Works

With the goal of analyzing attacker behavior, we presented LOGistICS, a multi-component system for safeguarding critical infrastructures. The system core is a new medium-interaction honeypot that simulates Modbus and S7Comm, respectively. These are some of the most commonly used protocols in industrial PLCs. This paper discusses the behavior of ICS actors when interacting with the honeypot, which is found to be less detectable by attackers than similar proposals, and indistinguishable from real PLCs when using automated attack tools. We tested how LOGistICS works by exposing it to the Internet and found that most of the requests made are not malicious. However, the people who make such requests do not always have the same behavior, which differs depending on the type of information they read. As for web crawlers, we figured out that the majority of them have the same behavior towards ICS devices; they essentially differ in the frequency of reading the fields that identify the device. Moreover, we found that besides periodic scanners (e.g. Shodan and Censys) who conclude their activity once the reconnaissance stage has been completed, there exist attacks trying to write data and control the PLC without knowing the victim, as if they already knew the necessary information. Our intuition is that these actors obtained them by looking for them indirectly from search engines, activity that would be classified by the intrusion detection systems as not suspicious (in the case of web crawlers are whitelisted). We collected several ideas about possible extensions as future work. For instance, we would like to leverage LOGistICS with SIEM-based technologies such as one based on the Elastic Stack.[14] The aim is to leave the honeypot running also in production environments (e.g. nuclear power plants), comparing it with that already obtained towards research environments. In this way, we can enhance our framework providing a more focused real-time alerting module. In addition, besides improving the extensibility and configurability of the honeypot, in the future we would like to add deception-technology modules and protocols such as *i) BACnet* (*Building Automation and Control*), which finds its application in ventilating, heating, access control, lightning, air-conditioning and fire detection systems; and *ii*) *DNP3* (*Distributed Network Protocol*), which is is widely adopted in the USA and Canada, and it is used by electric and water companies.

Acknowledgments

This work has been partially supported by: project “RACRA”- funded by Ricerca di Base 2018-2019, Univeristy of Perugia Project BLOCKCHAIN4FOODCHAIN: funded by Ricerca di Base 2020, Univeristy of Perugia Project DopUP - REGIONE UMBRIA PSR 2014-2020.

[14]Elastic SIEM: https://www.elastic.co/siem/.

References

[1] K. Stouffer, J. Falco, K. Scarfone, Guide to industrial control systems (ics) security, NIST special publication 800 (2011) 16–16.

[2] O. Analytica, Cyberattack on ukraine's power grid underlines risk, Emerald Expert Briefings (2022).

[3] A. Swales, et al., Open modbus/tcp specification, Schneider Electric 29 (1999).

[4] S. Bistarelli, E. Bosimini, F. Santini, A medium-interaction emulation and monitoring system for operational technology, in: ARES 2021: The 16th International Conference on Availability, Reliability and Security, Vienna, Austria, August 17-20, 2021, ACM, 2021, pp. 118:1–118:7.

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

[6] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, J. Schönfelder, A survey on honeypot software and data analysis, arXiv preprint arXiv:1608.06249 (2016).

[7] S. Bistarelli, E. Bosimini, F. Santini, A report on the security of home connections with iot and docker honeypots., in: ITASEC, 2020, pp. 60–70.

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

[9] K. Wilhoit, S. Hilt, The gaspot experiment: Unexamined perils in using gas-tank-monitoring systems, Trend Micro 6 (2015).

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

[11] S. Lau, J. Klick, S. Arndt, V. Roth, Poster: Towards highly interactive honeypots for industrial control systems, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 1823–1825.

[12] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, R. Beyah, Rethinking the honeypot for cyber-physical systems, IEEE Internet Computing 20 (2016) 9–17.

[13] J. You, S. Lv, Y. Sun, H. Wen, L. Sun, Honeyvp: A cost-effective hybrid honeypot architecture for industrial control systems, in: ICC 2021-IEEE International Conference on Communications, IEEE, 2021, pp. 1–6.

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

[15] E. López-Morales, C. Rubio-Medrano, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, G.-J. Ahn, Honeyplc: A next-generation honeypot for industrial control systems, in: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 2020, pp. 279–291.

[16] I. Modicon, Modicon modbus protocol reference guide, North Andover, Massachusetts (1996) 28–29.

[17] K. Ashton, et al., That 'internet of things' thing, RFID journal 22 (2009) 97–114.

[18] I. Modbus, Modbus messaging on tcp, IP Implementation Guide v1. 0a, North Grafton, Massachusetts (www. modbus. org/specs. php) (2004).

[19] P. Huitsing, R. Chandia, M. Papa, S. Shenoi, Attack taxonomies for the modbus protocols, International Journal of Critical Infrastructure Protection 1 (2008) 37–44.

[20] A. Kleinman, A. Wool, Accurate modeling of the siemens s7 scada protocol for intrusion detection and digital forensics, The Journal of Digital Forensics, Security and Law: JDFSL 9 (2014) 37.

[21] M. Rose, D. Cass, ISO Transport Service on top of the TCP-Version 3, Technical Report, RFC 1006, Northrop Research and Technology Center, 1987.

[22] D. Nardella, Snap7, 2018.

[23] A. Sentcha, Libnodave–exchange data with siemens plc, 2015.

[24] Siemens, Siemens block description, http://siemens-plc-programming.blogspot.com/p/load-memory-work-memory-blocks-in-user.html, 2010.

[25] E. Biham, S. Bitan, A. Carmel, A. Dankner, U. Malin, A. Wool, Rogue7: Rogue engineering-station attacks on s7 simatic plcs, Mandalay Bay/Las Vegas (2019).

[26] A. Kashtalian, T. Sochor, K-means clustering of honeynet data with unsupervised representation learning., in: IntelITSIS, 2021, pp. 439–449.

[27] M. J. Assante, R. M. Lee, The industrial control system cyber kill chain, SANS Institute InfoSec Reading Room 1 (2015).