A High-interaction Physics-aware ICS Honeypot for Industrial Environments

Kannan Kirishikesan, Gayakantha Jayakody, Ayesh Hallawaarachchi, Chandana Gamage

***Abstract*— Industrial Control Systems (ICSs) are control systems that automate and control industrial processes. ICSs have a high-security risk since most of them are connected to the Internet for remote monitoring and controlling purposes. Compromising ICS can disrupt critical infrastructure supplies, such as water supply, power supply, transportation systems, and manufacturing systems. Programmable Logic Controllers (PLCs) are special computers used in ICSs. Many PLCs do not have built-in security systems. Many ICS application layer protocols are not designed with security in mind. Therefore, external security systems are needed to protect ICSs from cyber-attacks. Identifying the vulnerabilities, malware, and attacking patterns is useful in designing defense-in-depth security systems for ICSs. Honeypots can be used for research purposes as a way of collecting data and can also be used to protect the systems from attackers. In this paper, we present a high-interaction physics-aware ICS research honeypot that has been extended to a production honeypot using Software Defined Networking.**

***Keywords*— Industrial Control Systems, SCADA, Honeypot, Snort IDS.**

I. Introduction

From the start of the 21st century, Industrial Control Systems have been an essential part of the industrial world. It has opened up considerable opportunities for the engineers as well as for the cybercriminals. Since most ICSs are connected to the Internet for remote monitoring and control purposes, there is a high-security risk for ICSs.

Supervisory Control and Data Acquisition (SCADA) is a system of software and hardware elements for monitoring, controlling and visualizing the real-time data generated in ICSs. PLCs are the basic components of SCADA. Many PLCs do not have built-in security systems and secure communication protocols to protect ICSs from cyber threats [1]. Therefore, external security systems are needed to protect ICSs from cyber-attacks. Several defense techniques exist, such as firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS), to defend ICSs against such threats, but they are not helpful in zero-day attack scenarios. Identifying the vulnerabilities, malware and attacking patterns in ICSs helps in designing and improving defense-in-depth security systems [2]. Honeypots are an effective way of collecting such types of real-world data.

Honeypots are computer systems left vulnerable on purpose for attackers to exploit. ICS honeypots are developed to behave similarly to the controllers used in the ICS domains such as PLCs. Honeypots can be classified according to their intent of existence or their level of interaction. Research Honeypots capture details and behaviour of the attack vectors, while production Honeypots can be deployed in ICS to protect ICSs from various types of attacks [3]. In this project, a high interaction physics-aware research honeypot is proposed that consists of a complete physical simulation of an ICS. The proposed research honeypot has been extended to a production honeypot using Software Defined Networking (SDN) [4].

Since the primary purpose of honeypots is to deceive cybercriminals, their effectiveness depends on their deceptive capabilities. A higher level of interaction and physics awareness are the main features that influence the deceptive powers of a honeypot. Physics awareness is the ability to replicate the timing delays and the actual physical behaviour of the system [5]. An attacker can quickly identify a honeypot that is not physics-aware as it does not obey the laws of physics or responds faster than an actual system.

The honeypot-based solution that is presented in this paper has a physical process simulation called HoneySim++ that was developed to make the system physics-aware along with a high-interaction ICS honeypot called HoneyPLC++ that has all the major ICS protocols implemented.

I. Background and Related Work

A. Classification of Honeypots

Honeypots can be classified into production and research honeypots based on their purpose. Production honeypots are used in production environments to mitigate the risk [6] [7]. Research honeypots are used to detect attack patterns of the attackers and produce a corpus of interactions between the attacker and the honeypot. This result can be further used to study the types of attacks that can occur in the ICS. It is a valuable method for identifying zero-day attacks.

Honeypots can be also categorized into low-interaction, medium-interaction, and high-interaction depending on their interaction level. Low-interaction honeypots are basic honeypots with only a few features of a real system. It

Correspondence: Kannan Kirishikesan (E-mail: kirishikesan.17@cse.mrt.ac.lk) Received:08.03.2023 Revised:22.03.2023 Accepted: 19.06.2023

Kannan Kirishikesan, Gayakantha K Jayakody, Ayesh S. Hallawaarachchi and Chandana Gamage are from University of Moratuwa, Sri Lanka. (kirishikesan.17@cse.mrt.ac.lk, gayanjayakody.17@cse.mrt.ac.lk, ayeshsandeepa.17@cse.mrt.ac.lk, chandag@cse.mrt.ac.lk)

DOI: https://doi.org/10.4038/icter.v16i2.7265

does not comprise a separate operating system, and attackers cannot gain complete access to the honeypot [8]. Their deceptive capability is relatively low, but it can be used as the foundation to build honeypots with higher levels of interaction. These honeypots are easier to build, maintain and deploy.

High-interaction honeypots have a greater deception capability than others by default since they include an operating system in their build. Due to their high-interaction functionalities, they act as a real system, capable of attracting attackers and collecting large amounts of information. This type of honeypots has increased difficulty in deployment as they require higher storage and processing resources.

Additionally, they require continuous monitoring to ensure that the honeypot is not compromised, hence becoming a vulnerability to the whole network. Medium-interaction honeypots are slightly more advanced than low-interaction honeypots [9] and have some features that can deceive the attackers. But they are not fully compromisable as high-interaction honeypots.

B. Comparison of Existing Honeypots

The first reported honeypot solution was the SCADA Honeynet Project [10] by CISCO critical infrastructure assurance group in 2004. It used the Honeyd framework to simulate the network services and python scripts to simulate the specific PLC devices. It was discontinued in 2005 due to its lack of deceptive capability.

Conpot is an active project by the honeynet project, first released in May 2013 [11]. It consists of SNMP, Modbus and HTTP protocols used in real PLC systems. Conpot is a honeypot built especially for ICS systems and facilitates easy extensibility, maintenance, and deployment after it has been configured properly. For multiple reasons, it is used as the baseline standard for developing various honeypot variants in the ICS domain. Conpot can be configured to include templates of different ICS devices including Siemens S7-200 PLC. It supports the major protocols in the ICS domain, such as HTTP, Modbus, SNMP, etc. Gaspot, Kampstrup smart meter and Gridpot for smart grids are some application-specific honeypots developed from Conpot.

CryPLH is a honeypot implementation that simulates Siemens Simatic 300 PLC using an Ubuntu environment [12]. All the network services are implemented in this honeypot, but with some deviations to the actual PLCs due to some Linux system file limitations. SNMP is implemented with a custom SNMP agent to respond to SNMP requests. And static web pages are created to serve HTTP and HTTPS requests. However, ladder logic capturing is not possible and physics awareness is also not implemented. Therefore, an experienced attacker may identify this as a honeypot.

Another popular honeypot for the PLCs is Honeyphy [5] developed at Georgia Institute of Technology in 2016. It consists of three major modules. Internet interface module, process models module, and device models module. Users interact with the Internet interface to access the services provided by PLC. Device models are there to simulate the PLC. Each device model can simulate any particular PLC. The Process modules are there to simulate the actual physical processes of the PLCs.

HoneyPLC [3] is a high interaction, extensible honeypot. It can deceive various reconnaissance tools like Nmap, Shodan's HoneyScore, Siemens Step7 Manager, PLCinject, and PLCScan [3]. It provides sophisticated service simulations for TCP/IP Stack, HTTP, SNMP and S7comm. Ladder logic injection capture was introduced in HoneyPLC, which was unavailable in the state-of-the-art honeypots. But HoneyPLC lacks physical interaction. The network stack is simulated using the Honeyd framework, which acts as the core of HoneyPLC.

TABLE I
COMPARISON OF ICS HONEYPOTS

Honeypot	Level of Interaction	Network Stack	Network Protocols	Ladder Logic Capture	Physics Awareness
SCADA Honeypot project	Low	Honeyd	Modbus HTTP FTP Telnet	None	None
Conpot	Low	None	S7comm HTTP SNMP	None	None
CryPLH	High	Linux Kernel Based	S7comm HTTP SNMP	None	None
HoneyPhy	Medium	None	DNP3	None	Yes
HoneyPLC	High	Honeyd	S7comm HTTP SNMP	Yes	None
Our Honeypot	High	Honeyd	S7comm HTTP SNMP Modbus	Yes	Yes – HoneySim++

Table I shows the comparison of ICS honeypots with respect to the primary factors which determine the deceptive capabilities of honeypots as per the details described above.

C. Limitations of Existing Honeypots

1) Physical process simulation: There are only a few high-interaction honeypots in the current literature, but none have a sophisticated physical process simulation. When attackers realize that the system, they are dealing with is not associated with any industrial control process, they give up their attempts and try to delete the footprints of their early activities. It is a disadvantage because the data about the extensive attack path of an attacker cannot be collected.

2) Extendability to Production Honeypot: Current literature lists many research ICS honeypots that are intended for use as a data collector. It is doubtful that an advanced attacker will attack a stand-alone research honeypot. An advanced attacker may collect information about the target system before launching an attack. Current studies show that most of the data collected by honeypots are basic reconnaissance steps. Honeypots need to be deployed along with a real production system to get more attention from the attackers. Current research honeypots do not provide a mechanism to connect them to a real industrial system.

II. System Design

A. Design Considerations

The research objective of this project is to develop a high-interaction honeypot addressing all the requirements to achieve greater deception capabilities through the following factors.

1. Responding as a real ICS
2. Integration of major ICS network protocols
3. High Interaction

HoneyPLC, a state-of-the-art honeypot, was chosen as the baseline for this project since it exhibited high deceptive capabilities through Shodan's Honeyscore. Shodan's Honeyscore, measured in the range of 0 to 1, is a metric used to determine a honeypot's deceptive capabilities. Low Honeyscore implies that the probability of the system being a honeypot is low and vice versa. An outcome of this research is the development of a research honeypot called HoneyPLC++ which was built by extending HoneyPLC with the following capabilities.

1) Modbus Protocol Integration: HoneyPLC has implemented the major ICS protocols except for the Modbus protocol, and integrating this missing protocol will complete the ICS network stack.

2) ICS Process Simulation with a Human Machine Interface (HMI): After entering the network, an attacker will make a reconnaissance attack to identify the controllers, devices and HMIs. The presence of the actual devices and the HMI will increase the chances of deceiving the attacker.

3) Physics Awareness: Implementing an actual ICS system or having a virtual ICS simulation are the two methods of implementing physics awareness in a honeypot-based system. Having a real duplicated ICS system just to be attacked is too expensive and dangerous. Therefore, virtual systems are preferred. Implementing a virtual ICS system that obeys the laws of physics can deceive the attacker more effectively.

4) Extensible to a Production Honeypot: Using SDN technology, it is possible to produce a honeypot that can be extended to a production honeypot.

B. System Design Architecture

The system is designed so that the honeypot-based system can be deployed parallel to the Industrial Control Systems. Fig. 1 shows the honeypot-based system's overall architecture, deployed in a production environment. A Virtual Private Network (VPN) endpoint exposes the system to the Internet, similar to many Industrial Control Systems. Then the traffic from the Internet is sent through an SDN switch. The SDN switch is deployed with an SDN controller and a Snort Intrusion Detection System. The SDN switch has the capability of dynamically changing the rules of routing when instructed by the SDN controller through OpenFlow rules. The traffic through the SDN switch is mirrored to the Snort, which will detect any suspicious packets and alert the SDN controller. The SDN controller can be programmed to change the packet forwarding rules of the SDN switch considering snort alerts. This will ensure that the traffic from the detected intruder will be redirected to the honeypot-based system. The internal network will have the following systems deployed parallelly.

- Human Machine Interface
- Several HoneyPLC++
- Industrial Control System

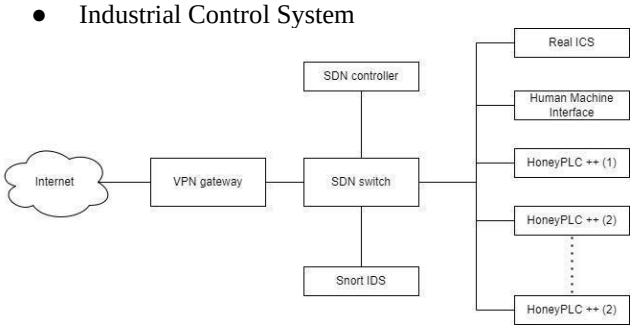

Fig. 1 Architecture of the system

III. Prototype Implementation

A. HoneyPLC++

HoneyPLC++ is developed leveraging HoneyPLC with Modbus protocol, water distribution simulation called HoneySim++, and a modified S7comm server. Two HoneyPLC++ variants are proposed. Depending on the requirement, variant 1, variant 2, or both can be placed in the system.

Both HoneyPLC++ variants use Honeyd to simulate the network stack, similar to HoneyPLC. Honeyd is a framework capable of simulating the network stack behaviour of any operating system that uses TCP/IP for communications. Honeyd was configured with correct OS details and configurations (Fig. 2) such that the tools such as Nmap can be deceived, and these tools will show that the configured HoneyPLC++ is an actual PLC, not a simulation. S7comm and HTTP requests coming to the Honeyd on ports 102 and 80 are redirected to the simulated S7Comm server and to the Lighttpd web server, respectively. As a new addition to HoneyPLC, a Modbus server is added to HoneyPLC++, and

the traffic coming to port 502 is redirected to the Modbus server.

```
create host1
set host1 personality "Siemens Simatic 300 programmable logic controller"
set host1 default tcp action block
set host1 default icmp action open
add host1 tcp port 102 proxy 127.0.0.1:102
add host1 tcp port 80 proxy 127.0.0.1:80
add host1 tcp port 502 proxy 127.0.0.1:1502
set host1 ethernet "00:1c:06:00:bc:37"
dhcp host1 on enp0s3
```

Fig. 2 Honeyd configuration file

In HoneyPLC, the Snap7 framework has been used to simulate the S7comm server. However, the S7comm server this framework provides can only work as the communication processor in a real PLC. Although it accepts the S7comm requests, it cannot handle those S7comm requests and respond accordingly. Furthermore, HoneyPLC used a modified version of the Snap7 framework, which can capture the ladder logic files. Anyone can upload ladder logic programs to HoneyPLC, which indicates that the PLC is not write-protected. But PLCs are protected with passwords in a real ICS. If someone uploads a malicious ladder logic program in HoneyPLC, the S7comm server cannot correctly execute the instructions in that program, and the system states will not be changed according to the ladder logic program. Hence a sophisticated attacker may identify that this is not a real PLC by monitoring the system state.

To rectify these shortcomings in the S7comm server, HoneyPLC was modified to build HoneyPLC++ variant 1. When someone tries to upload a malicious ladder logic, the S7comm server will always reply that the write functionality is password-protected, and the response will be an access error code. This type of reply will increase the deception capabilities of the honeypot. Though HoneyPLC++ variant 1 cannot capture any ladder logic programs, this behaviour will enable us to observe more advanced attack vectors.

HoneyPLC++ variant 2 allows attackers to upload ladder logic programs similar to HoneyPLC. Anyone who likes to analyse ladder logic programs uploaded by attackers can use this variant. Since HoneyPLC++ has a physical process simulation, analysers can expect more advanced malicious ladder logic programs to be uploaded than HoneyPLC.

Both HoneyPLC++ variants consist of HoneySim++, a physical process simulation of a water distribution plant.

B. *HONEYSIM++*

1) Overview of HoneySim++: An effective honeypot-based system in an industrial environment depends on having a proper physical process to deceive the attackers and keep them interested enough to reveal the attack vectors. HoneySim++, a water distribution system is developed with proper timing parameters for physical process simulation to address these requirements.

The simulated water distribution system shown in Fig. 3 has two main parts.

1. Water distribution - Simulates flow and the pressure distribution of the system in the pipelines, tanks and the high-rise reservoir in the system.
2. Chlorination - Simulates the chlorine concentration in the water in specific nodes of the system.

2) Components of HoneySim++: The water distribution system consists of three main components and four simulated processes. The components are,

- Sensors - These devices will sense the simulated processes' values and return the data through the Modbus protocol. The sensors added to the system are the hydro pressure sensor, flow meter sensor, and chlorine gas concentration sensor.
- Actuators - These devices will change the state of the simulated actuators, which can cause changes in the simulated physical processes. The implemented actuators are pumps and valves for water and chlorine.
- Physical Components - These are the physical parts of the system in which the sensors and actuators are fixed. A water well, pipelines, retention tanks, a storage tank, and a high-rise reservoir are implemented. Actuators and sensors are connected to these physical components. Physical properties associated with these components change with time according to the state of the actuators.

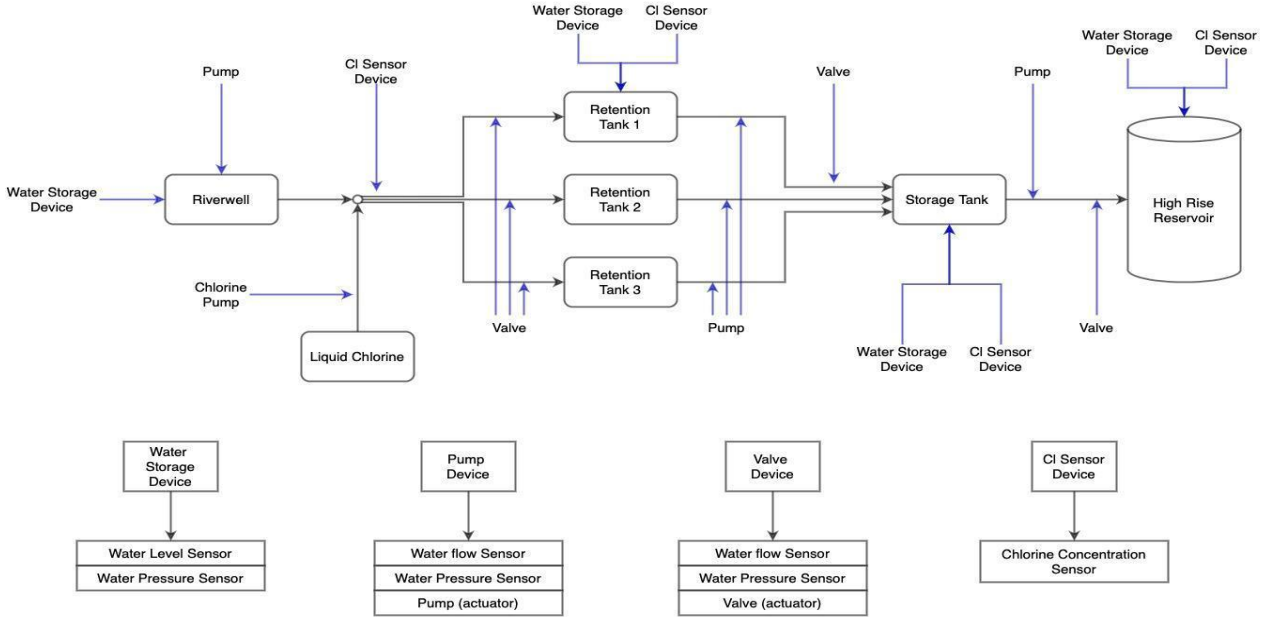

Fig. 3 HoneySim++ overview

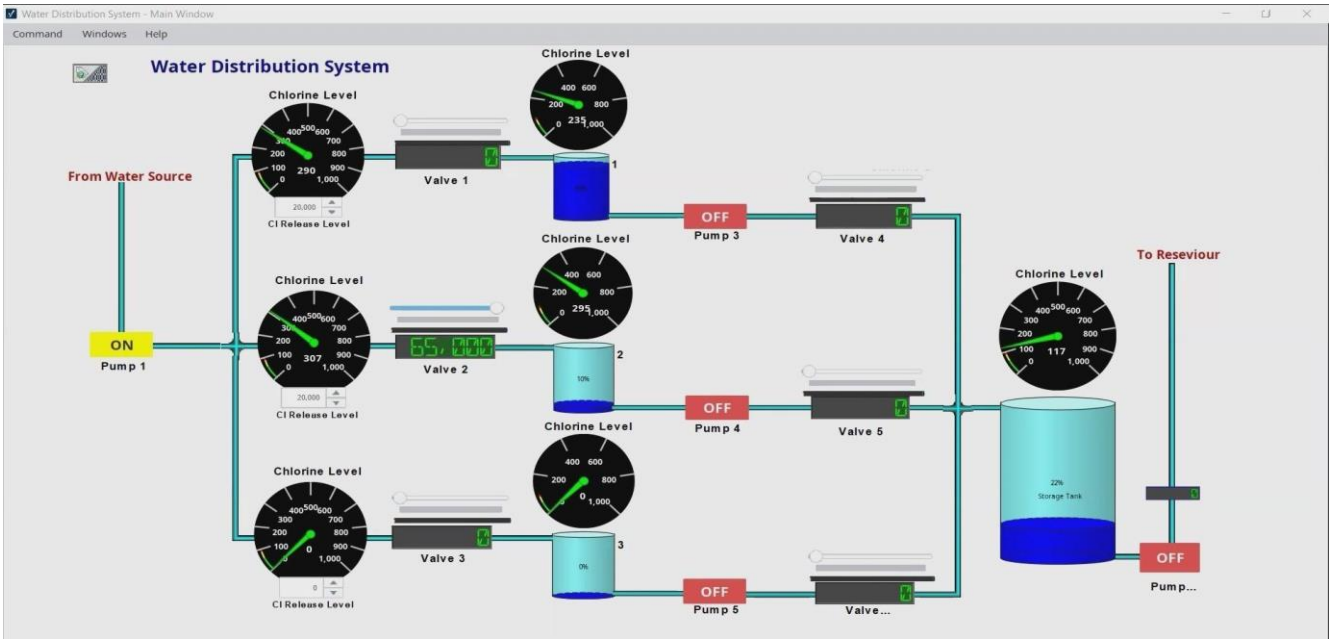

Fig. 4 HMI developed for the water distribution system

The physical Processes that are simulated in this system are,

- Hydro pressure
- Flow velocity in the system
- Chlorine gas concentration

3) Operation of HoneySim++: The operation of HoneySim++ is as follows. The physical components in the system are water storage tanks and pipelines. Pumps and valves connected between the pipelines and storage tanks control the water flow through the pipelines. In real water distribution systems, different devices contain several actuators and sensors, which are controllable using Modbus. Similar to those devices in real systems, several devices which contain sensors and actuators are simulated in Honeysim++. A Modbus server is attached to each device. Modbus commands are sent to the devices that contain the actuators to control the water flow and chlorine intake. Furthermore, threads that are running on the pipelines and storage tanks calculate how the water levels and chlorine concentration changes inside them. The mathematical model used to simulate the water levels, and chlorine concentration is described later.

4) Simulation of physical processes: Water level of the tanks and the chlorine concentration at the relevant points were simulated in HoneySim++.

Mathematical models simulate the change in water levels in the system and the time taken for the water to flow through the pipelines.

The water pumps are set to pump water at a particular flow rate, and the net water flow of the containers is obtained from the difference between the inflow and outflow. As per equation 1 the final water level (V) is calculated by adding the net water flow and the current water level (V_0). The net water flow is computed using input flow rate (F_{in}), output flow rate (F_{out}) and the cross-section area of the storage tank (A).

$$V = V_0 + (F_{in} - F_{out}) / A \quad (1)$$

The distance water flows in unit time (v_1) is calculated by dividing flow from the cross-section area of the pipeline, and the time (t) taken for water to flow through a pipeline is calculated by dividing pipeline length (l) by (v_1) as per equation 2.

$$t = l / v_1 \quad (2)$$

Chlorine is used to purify the water by removing the organic materials dissolved in the water. Chlorine rapidly reacts with the dissolved organic materials. Equation 3 from [13] is used to simulate the chlorine concentration (c(t)) changes in the water over time (t).

$$c(t) = c_0 \{ f e^{-k_R t} + (1-f)e^{-k_S t} \} \quad (3)$$

Here c_0 is the initial chlorine concentration and f, k_R and k_S are constants that depend on the total dissolved organic carbon (DOC) concentration of the water. The values for constants are taken from [13]. The final equation is shown as equation 4.

$$c(t) = c_0 \{ 0.35e^{-2t} + 0.65e^{-0.015t} \} \quad (4)$$

This equation simulates how chlorine concentration changes when flowing through the pipelines and when stored in storage tanks.

C. *Human Machine Interface*

Ignition SCADA software is used to build an HMI for the water distribution system. It is one of the most used SCADA software in the industry, and it is easy to configure and design. Fig. 4 is the screenshot of the HMI used in our simulation.

The primary purpose of adding an HMI to the honeypot is to easily identify any attacks done to the physical process simulation (HoneySim++). If an attacker changes any configuration detail in the PLC the normal behaviour of the water distribution plant will be disturbed and it can be easily detected using the HMI.

D. *Intrusion Detection Using Snort*

The system requires an Intrusion Detection System to identify any reconnaissance and penetrations. The use of an IDS simplifies the research work. Researchers can identify events of interest easily and go through the necessary logs when the IDS raises an alert. Security personnel get notified when the honeypot is deployed along with a production system and can act quickly to protect their systems. Snort, which is an open-source IDS, is used in this system. Rules to detect reconnaissance attempts, S7comm functions such as reading memory blocks and code injection attempts, Modbus register reading and writing attempts, etc., are added to Snort.

Snort IDS consists of a Modbus preprocessor which can be activated through the snort configuration file. This pre-processor is capable of identifying Modbus function codes and gives an easy way to define rules based on functions. For example, the rule in Fig. 5 generates alerts for write single coil attempts. Modbus preprocessor is not capable of identifying Modbus device identifications packets. Fig. 6 shows a custom rule written to identify Modbus device identification attempts considering the Modbus packet content. Custom rules have to be written for S7Comm packets as well because snort does not have a S7Comm preprocessor. Custom rule shown in Fig. 7 is to identify data block list attempts in the PLC.

Snort can be integrated with the SDN controller to assist the controller in making decisions about packet routing.

```
alert tcp [any,!192.168.8.128] any -> $HOME_NET 502 (msg:
"Modbus single coil write detected"; modbus_func:
write_single_coil; sid:1111003; rev:2; priority:1;)
```

Fig. 5 Snort rule for Modbus single coil write attempts

```
alert tcp [any,!192.168.8.128] any -> $HOME_NET 502 (flow
:from_client,established; content:"|00 00|",offset 2,depth
2; content:"|03|",offset 7,depth 1; msg:"Modbus Read Device
Identification detected"; classtype:attempted-recon; sid:
1111010; rev:2; priority:1;)
```

Fig. 6 Snort rule for Modbus device identification

```
alert tcp any any -> $HOME_NET 102 (content:"|32 07 00 00
01 00 00 08 00 04|"; offset:0; depth:10; content:"|00 01
12 04 11 43 01 00|"; offset:11; depth:8; msg:"S7 get list
of Data Blocks attempt"; sid:1111301; priority:1; rev:1;)
```

Fig. 7 Snort rule for PLC data block list using S7Comm

E. *Extendibility to a Production Honeypot*

A virtual network environment is configured as a prototype implementation using a GNS3 network emulator as shown in Fig. 8. Open vSwitch is used as the SDN switch, and an RYU controller is used as the SDN controller. Two HoneyPLC++ virtual machines are used: one as a real system and the other one as a honeypot. The same IP address is used for both HoneyPLC++ instances. Client 1 is assumed as a legitimate client, and client 2 is assumed as an attacker. RYU controller is programmed such that any network requests coming from the IP 10.0.0.10 is redirected to the real system, and packets from source addresses except 10.0.0.10 is redirected to the honeypot. Fig. 9 is the corresponding rule for this routing.

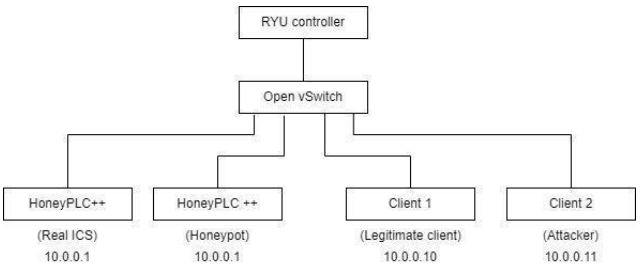

Fig. 8 Virtual network environment setup with GNS3

```
Match = eth_type=0x800, ipv4_src="10.0.0.10",
ipv4_dst="10.0.0.1"
Action = forward port 1
Priority = 2

Match = eth_type=0x800, ipv4_dst="10.0.0.1"
Action = forward port 2
Priority = 1
```

Fig. 9 A SDN rule used in the simulation

Similarly, we can add a list of IP addresses as an allowlist. Sometimes defining an allowlist can be difficult because of the difficulty in assigning static IP addresses for ICS administrators and operators. As a better approach, instead of having a predefined allowlist, the RYU controller can be programmed to accept alerts from Snort IDS and to decide which path the packet should follow.

IV. Prototype Evaluation

A. *Experiment Setup*

The experiment setup consists of three virtual machines with Ubuntu 20.04 LTS OS and the host machine with Windows 10 Enterprise OS. HoneyPLC++ is deployed in a virtual machine, HoneySim++ is deployed in another virtual machine that is only accessible from HoneyPLC++, and Snort IDS is running on the third virtual machine. The ignition SCADA system was deployed in the host machine.

B. *Reconnaissance*

As soon as the attacker enters the system, he will try out a few reconnaissance steps to get an overview of the system. Since this is a honeypot, the system should have good deception capabilities for reconnaissance attempts. The prototype implemented uses a HoneyPLC++ as the PLC and it was identified as a real device by an Nmap OS scan as depicted by Fig. 10. Fig. 11 shows that the Modbus server inside HoneyPLC++ responds with the device identification data for Modbus function 43 requests, which will lead the attacker to believe that this is a real PLC. These two identification mechanisms provide an idea that the attacker is dealing with an ICS. Mock data has been provided for this experiment, and real data can be provided for the Modbus server depending on the Modbus module in the PLC.

C. *Intrusion Detection*

Snort rules were added for ICMP packet detection, Modbus packet detection based on the function codes, and S7comm packet detection based on the functions. The system

```
gayan@ubuntuserver:~$ sudo nmap -O 192.168.8.100
Starting Nmap 7.80 ( https://nmap.org ) at 2021-12-06 12:35 UTC
Nmap scan report for 192.168.8.100
Host is up (0.020s latency).
Not shown: 999 closed ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 00:1C:06:F0:6F:99 (Siemens Numerical Control, Nanjing)
Device type: specialized
Running: Siemens embedded
OS CPE: cpe:/h:siemens:simatic_300
OS details: Siemens Simatic 300 programmable logic controller
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 6.87 seconds
```

Fig. 10 OS detection using Nmap

```
F:\FYP_Honeypot\ModbusCode>python client_test_fn_43.py
{0: b'SIEMENS', 1: b'SN-586', 3: b'https://www.siemens.com/global/en.html', 4: b'SIMATIC MODBUS/TCP CP',
 5: b'6AV6676-6MB00-6AX0'}
3
```

Fig. 11 OS Modbus device identification

```
++ [0] enp0s3
12/07-15:10:53.869561 [**] [1:1111007:2] "Modbus input registers read detected" [**] [Priority: 1] {
TCP} 192.168.8.128:13675 -> 192.168.8.15:502
12/07-15:10:53.882310 [**] [1:1111007:2] "Modbus input registers read detected" [**] [Priority: 1] {
TCP} 192.168.8.128:13676 -> 192.168.8.21:502
12/07-15:10:53.885044 [**] [1:1111007:2] "Modbus input registers read detected" [**] [Priority: 1] {
TCP} 192.168.8.128:13677 -> 192.168.8.24:502
```

Fig. 12 Alerts from Snort IDS

was tested to check whether relevant rules are triggered for the appropriate packets by sending various packets using ping, Nmap, PLCinject, and Modbus python scripts. Fig. 12 shows the snort alerts for Modbus input register read attempts.

D. Ladder Logic Uploading Attempts

HoneyPLC++ variant 1 was tested, which was reconfigured to deal with attackers who may upload malicious ladder logic files. The attackers will realize that the system is password protected and try more advanced techniques to deal with PLCs. Fig. 13 shows the packet capture when an attacker tries to upload a ladder logic program to the modified S7Comm server.

Fig. 14 is the response an attacker will get when he tries to upload a malicious ladder logic file to HoneyPLC++ variant 2. The file will be accepted as a valid input, but the state of the system will remain unchanged. This may alert the attacker that the system might be a honeypot.

E. Experiment with Physical Process Simulation

The realistic nature of the physical process simulation of the water distribution plant (HoneySim++) was tested by changing the state of the pumps, chlorine release pumps, and valves. When we change the chlorine release level of a chlorine pump, the chlorine level of the retention tanks gradually changes according to the release level. Initially, the chlorine release level is set to 20000, and the chlorine level of the retention tank will vary from 0.25-0.95 mg/L in the regular operation of HoneySim++. But any attacker can change the chlorine release level to a higher level by sending a crafted Modbus holding register write request to the relevant address of the Modbus server. The system was tested by increasing the chlorine release level to 65000, and the chlorine level of the retention tank gradually increased. Fig. 15 is a screenshot of the HMI after increasing the chlorine release level.

Attackers can change the valve opening levels too. The valve opening levels were changed by sending Modbus holding register write requests to the relevant address of the Modbus server and it was observed that the filling rate of the tanks varied according to the valve configuration.

Some critical functions like switching on and off the pumps are designed with basic fail-safe mechanisms in HoneySim++ such as, if the valve of the pump outlet or inlet is closed it cannot be switched on. We tested it by switching on a pump when the outlet valve of a pump is closed. We can observe that as soon as pump gets switched on it gets switched off. Attackers tend to believe that this is a real system since it has some control provided by the PLCs.

F. Limitations and Future Works

Many ICS components have limited network packet buffer sizes and cannot handle too many packets simultaneously. But HoneySim++ can handle lots of packets and simultaneous connections because it is deployed in a virtual machine. Therefore HoneySim++ does not behave realistically for Distributed Denial-of-Service (DDOS) attacks. This might cause the attacker to question the authenticity of the system that he is attempting to exploit.

HoneySim++ will be further developed to incorporate geographical conditions when simulating ICS systems. Suppose an attacker knows that the system is situated in a particular location. In that case, he might identify this system

s7comm

No.	Time	Source	Destination	Protocol	Length	Info
675	87.809769610	192.168.1.104	192.168.1.105	S7COMM	79	ROSCTR:[Job] Function:[Setup communication]
677	87.847753593	192.168.1.105	192.168.1.104	S7COMM	81	ROSCTR:[Ack_Data] Function:[Setup communication]
679	87.849045968	192.168.1.104	192.168.1.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
681	87.888151640	192.168.1.105	192.168.1.104	S7COMM	95	ROSCTR:[Ack] -> Errorcode:[0x8700]

```
Transmission Control Protocol, Src Port: 102, Dst Port: 41440, Seq: 50, Ack: 79, Len: 41
TPKT, Version: 3, Length: 41
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
S7 Communication
  Header: (Ack)
      Protocol Id: 0x32
      ROSCTR: Ack (2)
      Redundancy Identification (Reserved): 0x0000
      Protocol Data Unit Reference: 256
      Parameter length: 2
      Data length: 0
      Error class: Access error (0x87)
      Error code: 0x00
      [Error code: Unknown (0x8700)]
```

Fig. 13 Modified S7Comm error response for ladder logic inject

s7comm

No.	Time	Source	Destination	Protocol	Length	Info
60	4.861202689	192.168.1.104	192.168.1.105	S7COMM	79	ROSCTR:[Job] Function:[Setup communication]
62	4.896214831	192.168.1.105	192.168.1.104	S7COMM	81	ROSCTR:[Ack_Data] Function:[Setup communication]
64	4.897452208	192.168.1.104	192.168.1.105	S7COMM	85	ROSCTR:[Job] Function:[Read Var]
66	4.930928975	192.168.1.105	192.168.1.104	S7COMM	95	ROSCTR:[Ack_Data] Function:[Read Var]

```
Internet Protocol Version 4, Src: 192.168.1.105, Dst: 192.168.1.104
Transmission Control Protocol, Src Port: 102, Dst Port: 42102, Seq: 50, Ack: 79, Len: 41
TPKT, Version: 3, Length: 41
ISO 8073/X.224 COTP Connection-Oriented Transport Protocol
S7 Communication
  Header: (Ack_Data)
      Protocol Id: 0x32
      ROSCTR: Ack_Data (3)
      Redundancy Identification (Reserved): 0x0000
      Protocol Data Unit Reference: 256
      Parameter length: 2
      Data length: 20
      Error class: No error (0x00)
      Error code: 0x00
  Parameter: (Read Var)
      Function: Read Var (0x04)
      Item count: 1
  Data
    Item [1]: (Success)
```

Fig. 14 Successful S7Comm ladder logic inject response

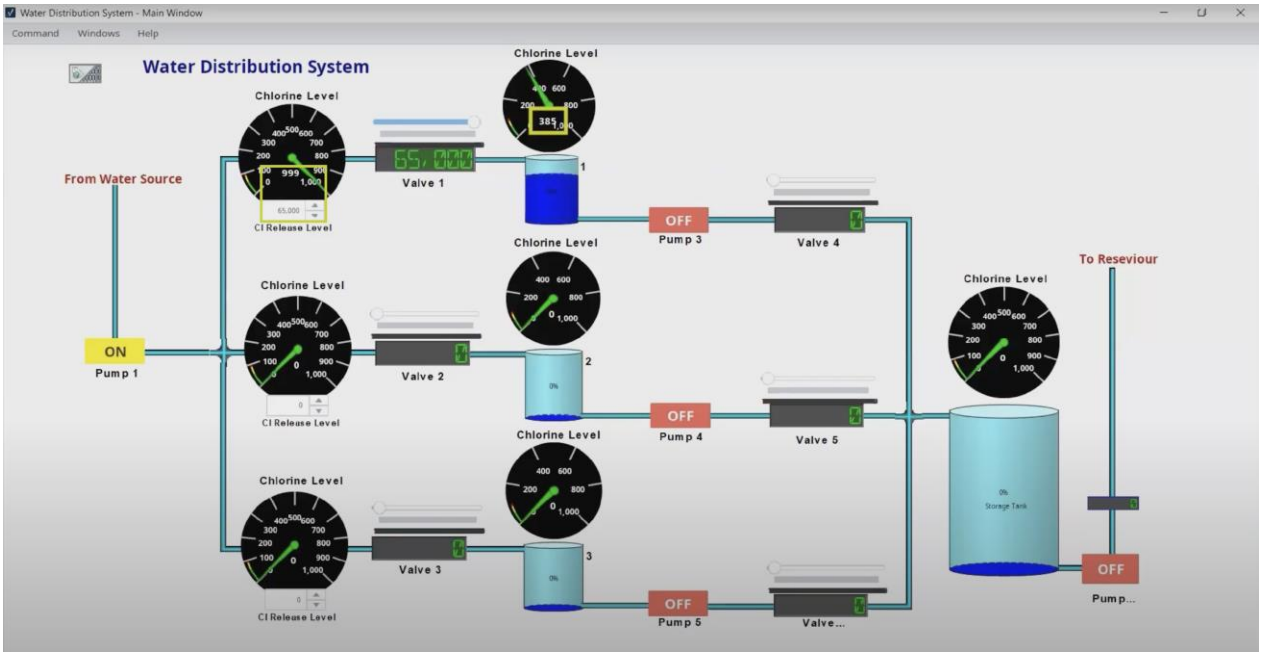

Fig. 15 HMI after chlorine release level increase attack

as a honeypot since the values generated do not correspond to the geographical conditions of the actual system. The resolution of the simulated physical processes should be adjusted to correspond to the resolution from the actual devices of the ICS, to increase the deceptive capability of the system.

There is no mechanism to inject a ladder logic program into the HoneyPLC++ variant 1 as it always an authentication error. HoneyPLC++ will be modified so that after a successful authentication, an attacker is able to inject a ladder logic program. But there must be a mechanism to run the uploaded ladder logic program inside the PLC simulation and allow the attacker to observe the changes after uploading the malicious ladder logic program.

The architecture of HoneySim++ is developed in a manner that allows incorporation of sensors, actuators and physical processes. If this system is to be implemented for a different ICS, HoneySim++ should be modified by adding the new sensors, actuators and physical processes relevant to the specific ICS.

An SDN controller program will be developed with a decision engine to decide whether a network request is legitimate or not based on the snort alerts. As per Fig. 1, several HoneyPLC++s can be designed to address various kinds of vulnerabilities. The SDN controller program can be developed to decide to which HoneyPLC++ the packet should be switched. Then the overall simulation will be deployed parallel to an actual water distribution plant, and the results will be analysed to identify new attack vectors.

V. Conclusion

Many ICS components and communication protocols are not designed with security in mind. The behaviour of threat actors should be identified first to safeguard ICS. This paper proposes a prototype research honeypot solution to identify the behaviour of the threat actors.

The prototype system provides the network protocols supported by an actual Siemens PLC - HTTP, SNMP, Modbus and S7comm. The honeypot includes a physical process simulation of a water distribution plant to achieve physics awareness. Since this honeypot is high-interaction, physics-aware and has most of the network protocols used in the ICS domain, the deceptive capability of this solution is significantly high. Tests performed on the system such as reconnaissance tests, intrusion detection, ladder logic injection and attacks on physical process simulation yielded positive results to verify the deceptive capabilities of the honeypot. The prototype honeypot can be deployed as a standalone system or along with an existing water distribution plant using Software Defined Networking

References

[1] G. Bonney, H. Hofken, B. Paffen and M. Schuba," ICS/SCADA security ¨ analysis of a Beckhoff CX5020 PLC," 2015 International Conference on Information Systems Security and Privacy (ICISSP), 2015, pp. 1-6..

[2] I. Hsien Liu, J. H. Lin, H. Y. Lai, and J. S. Li, "Extendable ICS honeypot design with modbus/TCP," in Proceedings of the International Conference on Artificial Life and Robotics, ICAROB 2022, 2022, pp. 287–290.

[3] E. Lopez-Morales et al., "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.

[4] H. Wang and B. Wu," SDN-based hybrid honeypot for attack capture," 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019, pp. 1602-1606, doi: 10.1109/ITNEC.2019.8729425.

[5]] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos and R. Beyah," Rethinking the Honeypot for Cyber-Physical Systems," in IEEE Internet Computing, vol. 20, no. 5, pp. 9-17, Sept.-Oct. 2016, doi: 10.1109/MIC.2016.103.

[6] M. H. Lopez and C. F. L. Resendez, "Honeypots: basic concepts, classification and educational use as resources in information security education and courses," in Proceedings of the Informing Science and IT Education Conference, 2008.

[7] K. Sadasivam, B. Samudrala, and T. A. Yang, "Design of network security projects using honeypots," J. Comput. Sci. Coll., vol. 20, no. 4, 2005, pp. 282–293.

[8] N. Kambow and L. K. Passi, "Honeypots: The need of network security," International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 5 (5), 2014.

[9] I. Mokube and M. Adams, "Honeypots: concepts, approaches, and challenges," in Proceedings of the 45th annual southeast regional conference, 2007.

[10] J. You, S. Lv, L. Zhao, M. Niu, Z. Shi, and L. Sun, "A Scalable High-interaction Physical Honeypot Framework for Programmable Logic Controller," in 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), Victoria, BC, Canada, Nov. 2020, pp. 1–5, doi: 10.1109/VTC2020-Fall49728.2020.9348483.

[11] A. Jicha, M. Patton and H. Chen, "SCADA honeypots: An in-depth analysis of Conpot," 2016 IEEE Conference on Intelligence and Security Informatics (ISI), 2016, pp. 196-198, doi: 10.1109/ISI.2016.7745468.

[12] D. I. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer, "Cryplh: Protecting smart energy systems from targeted attacks with a plc honeypot," in International Workshop on Smart Grid Security. Springer, 2014, pp. 181–192. 7,14

[13] D. C. Gang, T. E. Clevenger, and A. S. K. Banerji, "Modeling chlorine decay in surface water," Journal of Environmental Informatics, vol. 1, no. 1, 2015, pp. 21–27.