HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems

Marco Lucchese
Università degli Studi di Verona
Verona, Italy
marco.lucchese@univr.it

Francesco Lupia
Università della Calabria
Cosenza, Italy
francesco.lupia@unical.it

Massimo Merro
Università degli Studi di Verona
Verona, Italy
massimo.merro@univr.it

Federica Paci
Università degli Studi di Verona
Verona, Italy
federicamariafrancesca.paci@univr.it

Nicola Zannone
Eindhoven University of Technology
Eindhoven, The Netherlands
n.zannone@tue.nl

Angelo Furfaro
Università della Calabria
Cosenza, Italy
angelo.furfaro@unical.it

ABSTRACT

Industrial control systems (ICSs) are vulnerable to *cyber-physical attacks*, i.e., security breaches in cyberspace that adversely affect the underlying physical processes. In this context, honeypots are effective countermeasures both to defend against such attacks and discover new attack strategies. In recent years, honeypots for ICSs have made significant progress in faithfully emulating OT networks, including physical process interactions. We propose HoneyICS, a high-interaction, physics-aware, scalable, and extensible honeynet for ICSs, equipped with an advanced *monitoring system*. We deployed our honeynet on the Internet and conducted experiments to evaluate the effectiveness of HoneyICS.

CCS CONCEPTS

• **Security and privacy → Cyber-physical systems security**; *Intrusion/anomaly detection and malware mitigation*; *CPS security*.

KEYWORDS

Industrial control system, honeypot, physics-awareness, cyber-physical attack, attack monitoring

ACM Reference Format:
Marco Lucchese, Francesco Lupia, Massimo Merro, Federica Paci, Nicola Zannone, and Angelo Furfaro. 2023. HoneyICS: A High-interaction Physics-aware Honeynet for Industrial Control Systems. In *The 18th International Conference on Availability, Reliability and Security (ARES 2023), August 29–September 01, 2023, Benevento, Italy*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3600160.3604984

1 INTRODUCTION

Industrial Control Systems (ICSs) are physical and engineered systems whose operations are monitored, coordinated, controlled, and integrated by a computing and communication core [50]. They represent the backbone of *Critical Infrastructures* for safety-critical applications such as electric power distribution, nuclear power generation, and water supply.

As industrial organizations are increasingly connecting their operational technology (OT) network with the corporate networks to improve business and operational efficiency, ICSs are exposed to *cyber-physical attacks* [30, 40], i.e., security breaches in cyberspace that adversely affect the physical processes.

To defend ICSs from cyber-physical attacks, it is important to adopt a multi-layered approach. First, it is desirable to block cyber attacks before they reach the OT network. Indeed, most attacks against ICSs start by gaining access to the business network and then laterally moving to the OT network. Then, it is important to increase awareness of OT staff about cyber-physical attacks and teach them how to recognize these kinds of attacks. Last but not least, it is crucial to monitor and log remote connections to the OT network. This allows system engineers to detect attacks in their initial phase and to collect information about the techniques used by attackers in order to select appropriate mitigation.

Honeypots are computer security systems hosting virtual environments [24], which can emulate hardware and software devices and can be used to decoy attackers away from the real system, to educate staff, and to study attack patterns in order to select appropriate mitigation [43]. A system that consists of two or more honeypots is called *honeynet*. Although honeypots/honeynets for ICSs [13, 16, 19, 21, 22, 26, 35, 38, 41, 53, 55, 56] have significantly improved in the last years, they still have limitations regarding the following features that we deem essential to deceive attackers.

- *Level of interaction*: in order to deceive the attacker, ICS honeynets should be able: (i) to return accurate fingerprints of exposed OT components and the underlying industrial network (low-interaction), and (ii) to allow the attacker to interact with the honeypot (high-interaction) providing consistent emulation of physical feedback to attackers' actions.
- *Configurability*: including *extensibility*, to emulate different models of PLCs, and the possibility to adopt different industrial network protocols, depending on the context of use.
- *Scalability:* ICS honeynets should be able to reproduce the behavior of real-world ICSs, with possibly hundreds of devices.
- *Entry point*: to support attacks that may gain access to the honeynet either via a compromised VPN (used for remote engineers' accesses) or by exploiting OT devices directly exposed to the Internet. In the former case, ARP poisoning techniques can be exploited to perform non-trivial MITM attacks on the OT network.

- *Attack Monitoring*: ICS honeypots should have capabilities to collect data at the network and system level about attackers' strategies and provide valuable insights into the tactics, techniques, and procedures (TTPs) used by attackers, as well as help identify patterns and trends in malicious interactions.

Contribution. To address the above limitations, we propose HoneyICS, a high-interaction and physics-aware honeynet for ICSs (some preliminary concepts of HoneyICS appeared in [42]). HoneyICS emulates the OT network of an ICS. It supports a network of PLC and HMI honeypots rather than just a single PLC honeypot, like in most existing frameworks [33]. HoneyICS also supports high extensibility as it is able to emulate different brands of PLCs and ICS protocols. Our PLC emulation does not only include accurate emulation of common industrial network protocols but also allows an attacker to modify the value of PLC registers. Moreover, HoneyICS emulates physical plants connected to PLCs and, thus, can provide realistic feedback to attackers' commands. Last but not least, the attacker can gain full control of the network connecting PLCs and HMIs to mount non-trivial MITM attacks that can be accurately monitored by the honeypot administrator. The implementation of HoneyICS is based on containerization software that provides scalability and reconfigurability of the honeynet.

Outline. Section 2 provides background on OT networks. Section 3 presents design requirements, architecture, and attacker model for HoneyICS. Section 4 discusses the implementation of a non-trivial case study. Section 5 presents our experiments on a deployed instance of HoneyICS. Section 6 reviews existing ICS honeypots. Section 7 discusses security issues, extensions, and future work.

2 OT NETWORKS IN ICSs

Operational Technology (OT) networks connect devices, systems, networks, and controllers to operate and/or automate industrial processes. They include *programmable logic controllers* (PLCs) that control physical devices (sensors and actuators), and one or more *human-machine interfaces* (HMIs) to allow human operators to visualize and interact with the physical process. OT networks include two main sub-networks: the *supervisory control network* to connect PLCs and HMIs, and the *field communications network* to link the PLCs with the associated physical devices.

Programmable Logic Controllers. They are defined by the IEC 61131 standard [31] as "a digitally operating electronic system, designed for use in an industrial environment". PLCs have a simple ad-hoc architecture based on a central processing module (CPU) and further modules supporting physical inputs and outputs. The CPU executes the operating system of the PLC and runs a logic program defined by the user, called *user program*, written in one of the possible five languages allowed by the standard [31]. Additionally, the CPU is responsible for the communication with additional devices and manages the *process image*, i.e., a set of memory registers where all inputs (sensor measurements) and outputs (actuator commands) are copied. The user program operates on the process image, which is regularly refreshed by the CPU.

ICS Network Protocols. OT networks are interconnected through a variety of industrial protocols such as Modbus [15], DNP3 [27], EtherNet/IP [49], OPC UA [52], and S7comm [3]. In this paper, we focus on Modbus, which is the first and most used point-to-point industrial protocol located at the application layer. A Modbus transaction comprises a single query or response frame, or a single broadcast frame. Modbus TCP embeds a Modbus frame into a TCP frame. TCP/IP coordinators and workers listen and receive Modbus data via port 502. The protocol does not support security: an attacker could forge, drop or modify frames without being noticed.

3 OUR PROPOSAL

In this section, we propose HoneyICS, a high-interaction physics-aware ICS honeynet. We first describe our requirements for ideal ICS honeypots; then we provide the architecture underlying our honeynet together with the threat model.

3.1 Requirements for ICS Honeypots

ICS Honeypots have to face a number of challenges to provide the following features that we deem essential to deceive attackers.

Level of interaction. Honeypots and honeynets are usually classified based on the *level of interaction* they allow to the attacker. Low-interaction honeypots emulate one or more services with simple functions. While high-interaction ones emulate the behavior of real devices and are thus suitable to collect information about attackers' actions. An ICS honeypot should be able to simulate an industrial network connecting an arbitrary number of communicating PLCs, possibly supervised via HMI interfaces, and supporting observable and accessible network traffic involving PLCs and HMIs. Thus, an ICS honeypot should be not only able to return accurate fingerprints of the involved devices and ICS networks (*device and network emulation*) as is the case for low-level interaction ICS honeypots, but it should also allow an attacker to interact with the honeypot, for example, inspecting and modifying PLC registers, uploading malicious PLC code, inspecting and exploiting HMI interfaces, and basically gaining full control over the OT network. In addition, as pointed out in [36, 53], *physics-awareness* is a crucial ingredient to realize convincing and deceiving ICS honeypots; this means the attacker should receive consistent feedback from a (possibly simulated) manipulated physical process.

Configurability. The honeypot should allow an operator to change the attack surface exposed to an attacker in order to adapt to evolving attackers' exploit tools and techniques. Thus, the honeypot should be able to support different *industrial network protocols*, depending on the context of use. It should also be *extensible* to support the simulation of PLCs of different brands and models.

Scalability. To simulate real-world ICSs, the honeypot should *scale* to middle-size ICSs with hundreds of PLCs and HMIs of different kinds without affecting the performance.

Honeypot Entry Point. The attacker should gain access to the honeypot and its components either by compromising the VPN to which the honeypot is connected or by exploiting devices directly exposed to the Internet. In case the attacker is able to access the honeypot via the Internet, she can try to tamper with the exposed PLCs and HMI interfaces (eventually after a brute-force attack on their authentication). On the other hand, in case the attacker is able to compromise the VPN, she can do ARP poisoning in order to sniff network traffic on the supervisory control network and to set up MITM attacks between two PLCs or between a PLC and the associated HMI.

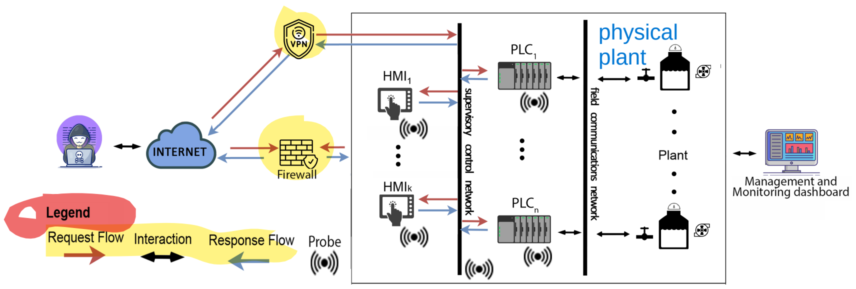

Figure 1: HoneyICS Architecture

Attack monitoring. The honeypot should provide functionalities to collect a rich set of data about the behavior exhibited by attackers. This data should include not only logs of network interactions to detect scanning attacks but also logs of system events capturing how attackers interact with the OS of the targeted ICS devices. Moreover, the honeypot should support real-time data analysis and visualization of information about the attacks as they occur and a posteriori data analysis of the logs to identify attack patterns. The extracted patterns can be used, e.g., to generate attack signatures to configure network and host intrusion detection systems.

3.2 Honeynet architecture

HoneyICS can emulate the key components of OT networks: PLCs, HMIs, communication networks, and a physical plant. These components can be implemented either using real physical devices or employing emulation and simulation software. Fig. 1 presents the architecture of our honeynet, putting together the components described below. Note that our honeynet can be configured to be accessible either via a (compromised) VPN or by exposing specific devices (PLCs or HMIs) on the Internet. The whole honeynet framework is managed and supervised via a management dashboard.

PLCs. Our honeypot can integrate both physical PLCs (i.e., hardware) and software PLCs. In the latter case, to support a realistic interaction with the attacker, HoneyICS combines the capabilities of both low-interaction and high-interaction physics-aware honeypots. Specifically, the *low-level interaction* is achieved by returning accurate fingerprints matching the profiles of the target PLC. At the same time, the *high-interaction* physics-aware honeypot emulates ICS network protocols supporting specific commands to modify PLC registers. This allows attackers to affect the physical process by reading/writing the values of PLC registers.

HMIs. The presence of HMIs allows us to support attacks where the attacker gains control of the HMI, for example, via password brute-force attacks or via phishing, and sends actuator commands directly to the PLCs through it.

Physical plant. Our architecture assumes the presence of underlying physical processes to realize *physics-awareness*, i.e., a convincing physical evolution of the compromised system observable by the attacker. Such processes could be implemented either using real physical devices or through real-time simulations done via classical tools.

Communication network. The previous components are connected via a communication network, which is divided into *supervisory control network*, connecting the PLCs among them and with HMIs (this network is devoted to exhibiting realistic network traffic), and *field communication network*, connecting the PLCs to the physical plant (transparent to the attacker).

Monitoring system. Collecting and tracking the activities performed by attackers or malware once they break into the honeypot is crucial for understanding their behavior and goals. Such activities involve the execution of various types of actions, e.g., dumping a binary file to inject malicious code or execute a new process, generation of network traffic, access to system resources, and so on. To achieve this goal, it is necessary to employ a specialized software component, ultimately acting like a probe, which can track the invocation of relevant system calls along with the associated parameters. Moreover, probes should identify sessions opened by an attacker to a specific honeypot and port. A key feature of such a probe is to operate without being detected by the malicious entity approaching the honeypot. In our honeynet, probes collect data involving access to each device (PLC or HMI) and the supervisory control network. Data collected by the probes is stored and sent to the management and monitoring dashboard for further processing.

Management & Monitoring Dashboard. The honeynet is managed via a web dashboard to simplify its configuration, deployment, and monitoring. It enables an operator to see what honeynets are currently running, add new honeynets, stop old ones, and see/analyze the data received from the probes dislocated in the honeynets. It also allows comparing what is displayed by an HMI interface with the real state of the physical process. This allows an operator to detect MITM attacks where an attacker feeds the HMIs with faked data that do not correspond with the actual evolution of the plant.

3.3 Attacker model

As shown in Fig. 1, our honeynet supports both Internet and VPN access. Specifically, our honeynet can be accessed either via the Internet, through exposed PLCs and/or HMIs, or via a (compromised) VPN. In the latter case, the attacker can take full control of the supervisor control network. In no case, the field communications network can be accessed by the attacker.

Once she has gained access, the attacker may fingerprint the target PLCs, using tools such as Nmap [28], to obtain basic system information (e.g., PLC model and brand, open and filtered ports, the services running on those ports, and the supported industrial protocols). In a subsequent step, the attacker may attempt to read and write PLC memory registers and upload and execute a malicious PLC user program. The attacker can observe the effect of the program execution either by examining the value of PLC registers or via a compromised HMI. Similarly, another possible attack vector is to fingerprint and then brute-force HMI interfaces to tamper with the physical state of the system by directly sending commands through the HMI interfaces. To this end, the attacker may exploit known vulnerabilities of the deployed (physical) PLCs and/or HMIs.

In case the attacker compromises the VPN under which the honeypot runs, she can: (i) sniff network traffic on the supervisory control network; (ii) set up MITM attacks between two PLCs or between a PLC and the associated HMI. In the latter case, the attacker may achieve a two-fold objective: on the one hand, she can manipulate PLC registers (e.g., those used to store actuator commands or sensor measurements) to bring the physical process into a compromised state; on the other hand, the attacker may transmit fake measurements to the corresponding HMI. These measurements may come from previous recordings made by the attacker on the genuine target system during an eavesdrop phase.

4 PROTOTYPE IMPLEMENTATION

4.1 Use case

As a use case, we consider a honeynet inspired by Lanotte et al.'s system [39], consisting of a network of three PLCs to control a simplified version of the *Secure Water Treatment system* (SWaT) [14], a scaled-down version of a real-world water treatment plant.

In the first stage, raw water is pumped in an 80 gallons tank T-201 via a pump P-101. A *valve* MV-301 connects tank T-201 with a *filtration unit* that releases the treated water in a second tank T-202 (with a capacity of 20 gallons). We assume that the flow of the incoming water in T-201 is greater than the outgoing flow passing through the (motor) valve MV-301. The water in T-202 flows into a *reverse osmosis unit* to reduce inorganic impurities. In the last stage, the water from the reverse osmosis unit is either distributed as clean water, if required standards are met, or stored in a backwash tank T-203 and then pumped back, via a pump P-102, to the filtration unit. We assume that tank T-202 is large enough to receive the entire content of tank T-203 at any moment (the capacity of T-203 is 1 gallon).

Each tank is controlled by a dedicated PLC. PLC-1 manages tank T-201. Intuitively, when pump P-101 is off, the water level in T-201 drops until it reaches its *low setpoint* (hard-coded in the memory register) $low1$. When this happens, the pump is turned on and remains active until the tank is refilled, reaching its *high setpoint* (hard-coded in the memory register) $high1$. Thus, for instance, if the pump is off, PLC-1 checks the water level of T-201, distinguishing between three possible states. If T-201 reaches its low setpoint $low1$, the pump is turned on, and the valve is closed. Otherwise, if T-201 is at some intermediate level between the low and high setpoint, PLC-1 listens for requests from PLC-2 to open/close the valve. Precisely, if PLC-1 gets an open request, it opens the valve, letting the water flow from T-201 to T-202; otherwise, if it receives a request to close the valve, it closes the valve. In both cases, the pump remains off. If the level of T-201 reaches its high setpoint $high1$, the requests for water from PLC-2 are served as before, but the pump is eventually turned off.

PLC-2 checks for the water level of tank T-202 and behaves accordingly. If the level reaches the low setpoint (hard-coded in the memory register) $low2$, PLC-2 sends a request to PLC-1, via a proper Modbus channel, to open valve MV-301, which lets the water flow from T-201 to T-202, and then returns. Otherwise, if the level of tank T-202 reaches the high setpoint $high2$, PLC-2 asks PLC-1 to close the valve via the same channel and then returns. Finally, if tank T-202 is at some intermediate level between $low2$ and $high2$, the valve remains open (respectively, closed) when the tank is refilling (respectively, emptying).

Finally, PLC-3 checks for the water level of the backwash tank T-203. If the level reaches the low setpoint $low3$, PLC-3 turns off pump P-102 and returns. Otherwise, if the level of T-203 reaches the high setpoint $high3$, P-102 is turned on until the entire content of T-203 is pumped back into the filtration unit of T-202.

The three PLCs are connected with an HMI to visualize the level of the three tanks and the current status of pumps and valves.

4.2 Implementation

In the implementation of our use case, we do not use any hardware or physical device, but rather relies upon existing simulation frameworks: Honeyd [46], HoneyPLC [41], OpenPLC [54], ScadaBR [1], and Simulink [45]. As for industrial network protocols we emulate *Modbus* [15]. As to scalability and reconfigurability, each component of our honeynet is deployed in a dedicated Docker container [17] running either Ubuntu 18.04 LTS or NGINX base images.

4.2.1 PLC containers. Our honeynet comprises three PLCs, which were implemented as software PLCs. To simulate a PLC, we created a container running both the *low-interaction* and the *physics-aware high-interaction* honeypot. The implementation of the low-interaction honeypot relies on Honeyd [46]. Honeyd allows us to simulate the TCP/IP stack of a target device such as PLCs. When a fingerprinting tool, such as Nmap, sends a TCP/IP request, Honeyd selects the proper fingerprint of the PLC from a fingerprint database and sends it back to the scanning tool. To support multiple PLC models, we integrated into Honeyd the PLC profiles provided by the HoneyPLC project [41]. HoneyPLC can currently answer fingerprint requests from the following PLC models: ABB PM554-TP-ETH, Allen-Bradley MicroLogix 1100, Siemens S7-300, S7-1200, and S7-1500. The HoneyPLC fingerprints are integrated into the Honeyd fingerprint database.

To implement the high-interaction physics-aware honeypot, we used *OpenPLC* [54], an open-source solution for PLC virtualization, compliant with the IEC 61131-3 standard [31]. The *network layer* of OpenPLC is responsible for establishing and maintaining network connections over Modbus by accessing the I/O image tables, i.e., the *PLC registers* storing, for instance, the current state of inputs (sensor measurements) and outputs (actuator commands).

To deceive an attacker, network requests should be routed to the proper honeypot. In particular, requests from scanning tools should be handled by Honeyd, whereas requests received via ICS network protocols, e.g., for register manipulation, should be handled by OpenPLC. For this reason, we integrated OpenPLC into Honeyd by using Honeyd's *subsystem virtualization* feature [46] to integrate new components via dynamic library preloading [44] by replacing the original networking functions of OpenPLC with Honeyd code.

Finally, to simulate a Modbus-based *supervisory control network*, we deployed in each PLC container a *broker* to connect a PLC with the other PLCs (the connection PLC-HMI via Modbus is already supported by OpenPLC and ScadaBR). When a PLC requires to transmit a message to another PLC, it writes the message in a dedicated register in its memory; the broker copies the value in this register into an associated register of the receiving PLC.

4.2.2 *HMI containers.*

To implement HMI components, we created a container and installed on it *ScadaBR*, an open-source drag-and-drop SCADA interface that can interact with several PLC brands. ScadaBR's *data sources* and *data points* are filled with data from the associated open-source PLCs via Modbus. ScadaBR's web interface is available on port 9090 and can be accessed using a web browser, via a weak authentication (admin/admin). Unlike software PLCs, HMI interfaces come with built-in communication support and do not need an ad-hoc broker for handling communications. Here, it is worth noting that we installed a version of ScadaBR (ver. 1.0CE) exposed to a *remote code execution* via an *authenticated arbitrary file upload* vulnerability (CVE-2021-26828). This vulnerability allows an authenticated attacker to execute arbitrary code on the targeted system by uploading a malicious .jsp file through the *view_edit.shtm* page. We will show how an attacker can exploit this vulnerability in Section 5.1.2 (Attack 2). The HMI container is equipped with a dedicated probe built on top of Aquasecurity Tracee [7], a runtime security forensics tool for Linux-based systems that leverages the extended Berkeley Packet Filter (eBPF) framework [9]. The probe is transparently integrated into the container and collects network traffic, system calls (we set a proper filter to trace only relevant calls), and kernel functions, as well as any changes to the filesystem. Additionally, the probe can capture any malware binaries dropped into the virtual environment by relying on Tracee's capabilities. Conversely, code injection and in-memory attacks are not fully supported by Tracee. Thus, we implemented an eBPF program alongside Tracee capable to detect the following fileless malware attacks: memfd_create-based, ptrace-based, ld_preload-based, pipe-based.

4.2.3 *Physical Plant container.*

We emulated the simplified water treatment plant described in Section 4.1 using *Simulink* [45], a framework widely adopted in industry and research to model, simulate, and analyze industrial control systems. The Simulink model runs inside a Debian-based Docker Container, which includes MATLAB Runtime and the needed toolboxes. In this container, we also implemented a second broker to simulate the *field communications network* connecting the physical plant with the PLCs. We accomplished this by using an existing C++ script called Simlink, which is distributed as part of the OpenPLC environment [54].

I stopped here:

4.2.4 *Supervisory Control Network.*

When an attacker reaches the honeynet, we must face two main challenges: (i) transparent redirection of the traffic generated by the attacker and involving containerized components; (ii) isolation of the components of the honeynet to avoid lateral movements on the supervisory control network in case an attacker compromises a device (PLC or HMI).

To achieve both tasks, each PLC component is logically connected to the supervisory control network through a *proxy* (see Fig. 2). Specifically, each PLC is assigned to a private `macvlan` docker network [10] along with a corresponding proxy component that sits in front of the PLC and acts: (i) as a *reverse proxy* for connections originating from both the Internet and the supervisory control network; (ii) as a *gateway* for connections originating from the PLCs. Note that proxies can communicate with each other via a secondary docker macvlan network they share. This is to support an under-control connection between two (possibly compromised) PLCs.

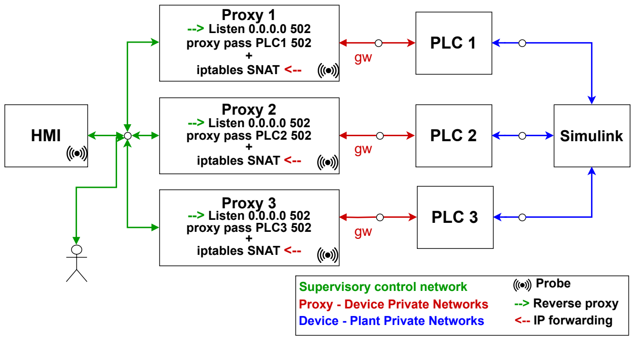

Figure 2: Implementation of the supervisory control network

We realized the proxy component supporting reverse proxying functionality using NGINX coupled with iptables post-routing commands for the gateway aspect. This allows us to change the source address of the connections originating from the corresponding PLC and, thus, properly route connections. Thanks to this proxying technique, our virtual environment is perceived by the attacker as a flat supervisory control network where she can set up *MITM attacks*. Proxies are transparent to the attacker as she sees them as real PLCs exposing their interfaces on the control networks.

In addition, the use of proxies allows us to shield, from entities connecting to the supervisory control network, details about the real nature (virtual/ physical) of the PLC, and, if necessary, to enforce security or filtering policies in the flowing traffic. Proxies are equipped with the same kind of probes seen in Section 4.2.2 for HMI containers. These probes allow us to capture all network traffic flowing to/from PLCs with a negligible impact on system performance.

4.2.5 *Management & Monitoring Dashboard.*

To facilitate the deployment, configuration, and runtime monitoring of HoneyICS, we developed a web dashboard that allows us to visualize the honeynets handled within the framework: each honeynet consists of one or more devices connected to each other. Each device can be managed through a device page. Here, we can add new devices in a honeynet either by providing a new docker image or by selecting a preexisting one; in this phase, we do port bindings to forward incoming network traffic on the host machine's port to the corresponding port of the container associated with the device. Once a containerized device is added to a honeynet, the "HTML page" option gives access to either the OpenPLC web interface or the HMI web interface, depending on whether the device is a PLC or an HMI. The "Start" option allows starting the container where the devices are deployed; similarly, the "Stop" option allows terminating the container.

All information gathered by the probes is stored in an Elasticsearch [8] data storage for further processing. To visualize the data collected in the Elasticsearch server, we embedded a Kibana [2] instance into the built-in dashboard that provides a comprehensive view of attackers' activities against each honeypot. This includes the list of attacks targeting each honeypot, the origin of these attacks, and the type of malware used. Moreover, we leverage the Alienvault OTX [6] API to enrich the collected data with MITRE ATT&CK [11] TTPs and employ VirusTotal [12] API for the classification and analysis of any captured malware. The dashboard provides a visualization of the detected attack patterns in *real-time*, including the most frequently targeted ports and services, the total number of connections established, their duration, and the number of unique attackers. It also features a geographically enriched attack map, displaying the origin coordinates of attacks as they occur. All visualizations are interactive and allow for advanced filtering mechanisms.

Additionally, the dashboard allows us to visualize the Simulink simulation of the honeynet in comparison with its HMI interface(s). Thus, the operator can visualize MITM attacks where the information provided by the HMI interfaces is somehow compromised.

5 EVALUATION

We performed a set of experiments to evaluate the *effectiveness* of HoneyICS. In particular, we evaluated the design choices underlying HoneyICS using both a local deployment, for testing our framework against network scanning tools and full-of-concepts attacks, and a remote deployment on an Italian Academic network, for evaluating HoneyICS in a real environment.

5.1 Evaluations of a local deployment

Setting. We deployed HoneyICS on a dedicated Dell PowerEdge T620 server with the following specifications: Intel Xeon E5-2640 v2 with 16 CPUs and 128 GB of RAM, running VMware ESXi. Three virtual machines were created. The first VM runs Docker with both the PLC, the HMI, and the proxy containers, and is equipped with 4 vCPUs, 8 GB Ram, 200 GB disk, and 5 Network adapters. The second VM runs Docker with the Matlab Simulink container and has 4 vCPUs, 8 GB Ram, 200 GB disk, and 1 Network adapter. The third VM runs the management and monitoring dashboard, including Elastic and Kibana, and features 4 vCPUs, 12 GB Ram, 500 GB disk, and 1 Network adapter.

5.1.1 *Fingerprinting Experiment.*

The goal of this experiment is to test the emulation effectiveness of HoneyICS against scanning tools.

Methodology. We scanned the IP addresses of the exposed devices of our prototype (i.e., the proxies associated with the three PLCs and HMI). We were connected to the honeynet via a VPN and used two different tools: Nmap (vers. 7.80) and PLCscan. Nmap [28] is a reconnaissance tool that allows the attacker to scan a network of machines to detect, with a given confidence, the operating system of each machine and the services running on them. In particular, we run Nmap's aggressive OS guesses using command *nmap -O –osscan-guess target-ip*, where the "-O" option and "-osscan-guess" enable accurate OS detection. PLCScan [34] is a reconnaissance tool designed to scan PLC devices based on either S7comm or Modbus. It returns a list of PLC information, including basic hardware, serial number, name of the PLC, model, and firmware version.

Results. Replicating the system scan described in [41], our results showed that Nmap's aggressive OS guesses indicated that the hosts were actual PLCs with an accuracy ranging from 98% for the Micrologix 1100, and 99% for the Siemens Simatic 300 and 1200, As expected, by scanning the HMI with Nmap we get that the TCP port 9090 is open and running Apache Tomcat/Coyote JSP engine 1.1. Finally, when applying PLCscan to the PLCs we get for each PLC two Modbus units on port 502 and identified by their IDs.

5.1.2 *Attack Simulation Experiment.*

We test the capabilities of HoneyICS in supporting the attacker model of Section 3.3. To this end, we simulated three attacks inspired by a series of annual cyber-physical defense exercises on the SWaT system [14], referred to as *Critical Infrastructure Security Showdowns* (CISS) [25].

Here, we recall that in the implementation of our use case, we adopted Modbus as the underlying ICS protocol. Modbus maps the temporary memory of a PLC program to four different kinds of registers: (i) *discrete output coils*; (ii) *discrete input contacts*; (iii) *analog input registers*; (iv) *analog output holding registers*; the latter registers are also used as *general memory registers* of different sizes (16-32-64 bits). The commands used to manipulate these registers are called *function codes* and allow for: read/write access on both coil registers and holding registers, and read-only access on both discrete input registers and input registers. A Modbus frame message contains the address of the intended receiver, the command the receiver must execute, and the data needed to execute the command. We recall that Modbus TCP typically listens and receives data via port 502. Last but not least, we recall that the protocol does not support security: an attacker could forge, drop or modify frames without being noticed.

Methodology. We executed three different attacks to cover the attacker model of Section 3.3. The proposed attacks are distinguished by the following aspects: *attack vector*, *intended attacker's goal*, *specific targets* of the attack, and *observability/stealthiness* when looking at the SCADA HMI interfaces. We evaluate our simulations using our management & monitoring dashboard to observe at the same time both the HMI interface and the runtime evolution of the simulated system under attack. Moreover, we investigate the logs collected from the probes distributed in the honeynet to provide further evidence of the ongoing malicious activity step by step.

Attack 1: In this attack, we assume that both PLCs and HMIs are exposed (and hence accessible) on the Internet. Our first attack is a *DoS* on pump P-101 governed by PLC-1 to achieve *tank overflow* of tank T-201. In particular, when tank T-201 reaches the high setpoint $high_1$ (80 Gal.), the attacker forces the pump to remain on. This is achieved by the attacker using a script that keeps transmitting a Modbus packet, with function code *read analog input register* 0x04, on port 502/TCP, to read the PLC register associated with the level of tank T-101. When the level reaches the value $high_1$, the script keeps transmitting a different Modbus packet, with function code *write single discrete output coil* 0x05, on port 502/TCP, to manipulate the PLC register associated with pump P-101. As a consequence, independently from the state of valve MV-301, the attacker will achieve *tank overflow* of tank T-201.

Attack 2: In this attack, we assume the HMI is exposed on the Internet (recall that the HMI container hosts ScadaBR, version 1.0CE). The attacker exploits the *authenticated arbitrary file upload* vulnerability CVE-2021-26828 of ScadaBR to gain complete control of the HMI and carry out a MITM attack aiming to drag the physical system into a deadlock. By exploiting CVE-2021-26828 combined with the use of default credential vulnerability, the attacker gains a web shell on the container. This, in turn, grants remote command execution, which is exploited to upload a statically linked binary that does the following: (i) it keeps transmitting a Modbus packet, with function code *read discrete output coil* 0x01, on port 502/TCP, to read the PLC register associated with valve MV-301, and (ii) sends a second Modbus packet, with function code *read analog input register* 0x04, on port 502/TCP, to read the PLC register associated

Table 1: Attacks detection and classification

	Log source (container, probe)	Payload	MITRE ATT&CK Technique	Description of the attack
attack 1	Network (proxy 1, Tracee)	e984b820501400000cb40000	active scannings	identification of the Modbus protocol
	Network (proxy 1, Tracee)	000100000000601040000000 1	monitor process state	keep sending "read analog input register" packets
	Network (proxy 1, Tracee)	00020000000601050000ff00	DoS	flooding of “write single discrete output coil” packets
	Network (HMI, Tracee)	POST/ScadaBR/login.htm username=admin&password=admin	default accounts	POST request to log into the HMI using default credentials
	Network (HMI, Tracee)	GET/ScadaBR/views.shtm	monitor process state	GET request to display the HMI view
attack 2	Network (HMI, Tracee)	76bf1c4350140000c02d0000	active scannings	attempts to identify the HTTP port of the HMI
	Network (HMI, Tracee)	POST/ScadaBR/login.htm username=admin&password=admin	default accounts	POST request to log into the HMI using default credentials
	Network (HMI, Tracee)	GV_369755 <%@page import="java.lang.*"%>	exploit public-facing application	POST request to /ScadaBR/view_edit.shtm using the shell as payload
	Network (HMI, Tracee)	GET/ScadaBR/uploads/1.jsp?ipaddress=...	exploitation of remote services	connection to the attacker’s machine through webshell
	System (HMI, Tracee)	eventName: execve, args: /bin/whoami, process: tomcat, pwd: /opt/tomcat6/apache-tomcat-6.0.53/webapps/ScadaBR/	command and scripting interpreter	commands to acquire knowledge on the infiltrated system
	System (HMI, Fileless)	eventName: recv, sockfd: curl -O http://attacker_ip:1337/script.sh|bash	command and scripting interpreter	a remote bash script executed from hmi’ memory to download a statically linked GoLang binary
	Malware (HMI, Tracee)	1c8252872ab34a3141ed56aa2f16051b	upload malware	A GoLang binary that implements ARP Spoofing is dropped into the HMI container
	System (HMI, Tracee)	7ed300000006010100000001	monitor process state	sends a "read discrete output coil" and a "read analog input register" packets
	System (HMI, Tracee)	eventName: sockfd: "ARP reply sent: PLC2 is-at attackerMAC"	adversary in the middle	the attacker tells PLC2 she is PLC1 and drops all the packets
attack 3	Network (proxy 3, HMI, Tracee)	e984b820501400000cb40000	active scannings	attempts to identify the HMI and the PLC3
	Network (proxy 3, Tracee)	000100000006010400000001	monitor process state	keep sending "read analog input register" packets to PLC3
	Network (HMI, Tracee)	eventName: sockfd: "ARP reply sent: PLC3 is-at attackerMAC"	adversary in the middle	the attacker tells the HMI she is PLC3
	Network (proxy 3, Tracee)	00020000000601050000ff00	DoS	flood PLC3 with "write single discrete output coil" packets
	Network (HMI, Tracee)	POST/ScadaBR/login.htm username=admin&password=admin	default accounts	POST request to log into the HMI using default credentials
	Network (HMI, Tracee)	GET/ScadaBR/views.shtm	monitor process state	GET request to display the HMI with the fake view

with the level of tank T-101. Finally, (iii) when the level of the tank approaches the value $high_1$ (80 Gal.) and the valve is still closed, the attacker drops all Modbus packets transmitted from PLC-2 to PLC-1 to intercept all requests to open valve MV-301. As a consequence, the valve remains closed and, in the long run, the system reaches the following *deadlock state*: tank T-201 is full (without overflow), and tanks T-202 and T-203 both face underflow.

Attack 3: We assume that our honeynet is under a compromised VPN and simulate a *stealthy DoS* attack on pump P-102 governed by PLC-3. When tank T-203 reaches the low setpoint low_3 (0 Gal.), the attacker forces the pump to remain on. This is done using a script that keeps transmitting a Modbus packet, with function code *read analog input register* 0x04, on port 502/TCP, to read the PLC register associated with the level of tank T-203. When the level reaches the value low_3, the script keeps transmitting a different Modbus packet, with function code *write single discrete output coil* 0x05, on port 502/TCP, to manipulate the PLC register associated with pump P-102. Thus, P-102 keeps working in the absence of water and eventually breaks. In this case, we assume the attacker aims to remain *stealthy* and set up a *MITM attack* by sending to the HMI interface Modbus packets saying that the pump is off (again, function code *write single discrete output coil* 0x05) when actually it is on.

Results. We discuss the impact of three attacks on the physical processes underlying our honeypot implementation along with the security events captured by the employed monitoring systems.

The impact of attack 1 on the system can be observed by the security analyst via the *management dashboard.* Indeed, the HMI shows tank T-201 at the maximum level while pump P-101 is still on. Moreover, the graphical representation of the simulation of tank T-201 shows its level going well above the maximum, denoting the overflow of the tank. The attacker can observe the state of the system by forcing the authentication into the HMI. The three main steps of the attack: measurement readings of PLC1, flooding of the actuator associated to pump P-101, and HMI access, have been detected by the monitoring system, as depicted in the first part of Table 1.

The physical deadlock of the tank system induced by attack 2 can be observed by the security analyst by looking at the HMI and/or the graphical representation of the levels of the three tanks in the management dashboard. Again, the attacker can use the compromised HMI to observe the system in the desired deadlock state. The security events logged by the monitoring system, including the exploitation of the vulnerability of ScadaBR and the consequent upload of the malware to achieve a MITM between PLC1 and PLC2, can be found in the middle part of Table 1.

Unlike the previous attacks, attack 3 is stealthy due to the MITM targeting the HMI. Thus, the security analyst can observe via the web dashboard an inconsistency between the real evolution of the system, as provided by the graphical representation of the simulated process, and what is shown by the HMI. The main steps of the attack, including the MITM between the HMI and PLC3, the measurements readings of PLC3, the flooding of the actuator associated with pump P-102, and the HMI access, have been detected by the monitoring system, as reported in the third part of Table 1. Note that due to the ARP spoofing, the HMI is now retrieving information from a server operating on port 502 of the machine of the attacker, who recorded the measurements during the initial reading phase.

Table 2: Requests and interaction received per device

	PLC1 (Modbus)	PLC2 (Modbus)	PLC3 (Modbus)	HMI (HTTP)
Well-formed requests	1774	1659	1666	22108
Bad requests	64	48	49	0
Well-formed interactions	963	896	905	7798
Bad interactions	64	48	49	0

5.2 Evaluations of a remote deployment

Settings. We exposed our honeynet on four endpoints across the Italian Academic Consortium GARR. The settings of this deployment are the same presented in Section 5.1 for the local deployment. Our honeynet was exposed to the Internet for a period of three months, and all interactions were logged using our web dashboard.

5.2.1 Shodan's Honeypot tag Experiment. The goal of this experiment is to test whether Shodan recognizes HoneyICS as a honeynet.

Methodology. Given an IP address, Shodan may assign it an *honeypot* tag. To verify whether Shodan recognized our honeynet, we checked for the "honeypot" tag on the four IP addresses used in the deployment. Other possible tags assigned by Shodan are: ICS, IoT, database, cloud, and self-signed (for certificates), among others. These tags provide additional information about the device or service associated with the IP address.

Results. Shodan scanned the HoneyICS IP addresses and included them in the search results within a week. None of the four devices exposed were classified as honeypots by Shodan. Instead, the three IP addresses hosting (the proxies of) the PLCs were marked with the tag "ICS", which indicates genuine ICS devices. However, no tag was assigned to the IP address hosting the HMI, as it exposes only a TCP port running the HTTP protocol.

5.2.2 Internet Exposure Experiment. This experiment aims at evaluating the level of interaction and attraction of our honeynet.

Methodology. We logged all connections of external actors with the exposed devices of the honeynet (PLCs and HMI), distinguishing between: well-formed (WF) and bad requests. Well-formed requests match the protocol assigned to the destination port, while bad requests do not (e.g., HTTP requests to the Modbus/TCP 502 port). We also reconstructed WF and bad interactions with the honeynet by aggregating requests within the same socket connection. Specifically, interactions are well-formed if they consist of only WF requests, while bad interactions comprise only bad requests; we did not observe any interaction comprising both WF and bad requests. To gain more insights, we also used GreyNoise [5] to analyze the IP addresses from which WF requests originated. In particular, we derived the autonomous systems to which the IPs belong, distinguishing their requests based on the used protocol (Modbus or HTTP). According to GreyNoise, benign IPs are associated with legitimate entities and have opt-out functionality; malicious IPs are IPs that exhibited harmful behaviors; unknown IPs do not meet the criteria for benign or malicious IPs and include Internet scanning services.

Results. Tables 2 and 3 report the results of our analysis. From Table 2, we observe that Modbus requests are equally distributed between the three PLCs, indicating that the three profiles have the same attractiveness. The Modbus requests include: ENCAP _INTERFACE_TRANSPORT, READ_COILS _EXCEPTION, READ _HOLDING_REGISTERS and REPORT_SLAVE_ID. We observed interesting malicious Modbus requests originated from DigitalOcean, for instance, using the Stretchoid scanner. Notably, DigitalOcean is also the autonomous system associated with the largest number of malicious IPs (201), with its IPs classified as malicious in more than 72% of the cases. On the other hand, the number of HTTP requests is sensibly higher compared to the one of Modbas requests (22108 vs. 5099). The vast majority of HTTP requests (93.4%) originated from LG DACOM Corporation, STARCLOUD GLOBAL PTE. LTD, and Censys. Notably, our monitoring system detected 21 successful login attempts to authenticate in the HMI interface. Among them, an attacker employed a VPN service hosted by Quadranet Inc.; in this case, the user agent indicated the use of a browser masquerading as Mozilla Firefox, running on Android 6.0 (Nexus 5). Upon authentication, the attacker explored the HMI graphical interface (/ScadaBR/dwr/engine.js" and "/ScadaBR/dwr/interface/MiscDwr.js"), presumably to ascertain the version of the Direct Web Remoting library.

Table 3: Main autonomous systems per WF requests

Autonomous system	State	WF requests			Unique IP			
		total	HTTP	Modbus	total	mal.	ben.	Unkn.
LG DACOM Corporation	KR	11198	11198	0	1	0	0	1
STARCLOUD GLOBAL PTE., LTD.	US	7045	7045	0	1	0	0	1
Censys, Inc.	US	3076	2400	676	154	0	93	61
DigitalOcean, LLC	US	1710	161	1549	277	201	28	48
Akamai Connected Cloud	US	746	370	376	56	16	40	0
Hurricane Electric LLC	US	522	0	522	37	0	37	0
CariNet, Inc.	US	514	2	512	11	2	8	1
Amazon.com, Inc.	US	443	133	310	28	3	0	25
INTERNET MEASUREMENT	GB	399	31	368	137	0	137	0
Google LLC	US	301	14	287	146	5	141	0
Delis LLC	US	152	152	0	2	2	0	0
Cogent Communications	AT	90	12	78	9	0	9	0
IP Volume inc	NL	81	26	55	8	3	5	0
FASTNET DATA INC	US	79	79	0	1	0	0	1
UCLOUD INFORMATION TECH.	HK	43	43	0	15	8	0	7
AgotoZ HK Limited	HK	21	21	0	15	14	0	1

6 COMPARISON WITH RELATED WORK

Table 4 summarizes and compares related work with HoneyICS according to the desired requirement for ICS honeypots (cf. Sec. 3.1).

Level of interaction. Current approaches mostly provide limited functionality related to TCP/IP stack simulations, as well as native ICS network protocols. This poses serious limitations in the actions an attacker can perform within the honeypot and, thus, in the understanding of adversarial interactions and malware. Only Hilt et al. [29] and Antonioli et al. [19] support non-trivial MITM attacks between PLCs and/or HMIs; more limited forms of MITM attacks, between PLCs and their plant, can be simulated in [13, 20, 26, 53]. Only [16, 18, 20, 22, 26, 29, 48] explicitly support some form of register manipulation, and only [18–21, 29, 48, 53] explicitly support some form of HMI manipulation. As regards *physics-awareness*, only the works in [13, 18, 19, 26, 29, 48, 53] provide some form of simulation of the underlying physical industrial processes. Moreover, only HoneyPLC [41] supports the upload of malicious user programs, although the injected code is only stored but not executed. While capturing injected code is important to support malware analysis, the execution of the injected code along with consistent physical feedback is crucial to deceive the attacker. HoneyICS has been designed to provide attackers with a level of interaction similar to the one of a real ICS. To this end, HoneyICS combines a low-interaction honeypot with a high-interaction one to simulate PLCs and integrate an interactive physical process. Moreover, it implements the

Table 4: Comparison with other ICS honeypots

✗= Not supported ◒= Partially supported ✓= Fully supported

Honeypot	Level of Interaction					Configurability		Honeypot Entry Point	Attack monitoring	
	ICS Network simulation	Physics aware	PLC registers	HMI	MITM	ICS protocols	Extens.		Data Collection	Data Analysis
SCADA Honeynet [55]	◒	✗	✗	✗	✗	Modbus	◒	Internet	–	–
Conpot [38]	◒	✗	✗	✗	✗	Modbus, S7comm, BACnet, EtherNet/IP	◒	Internet	network traffic	–
Dipot [32]	◒	✗	✗	✗	✗	Modbus, S7comm, BACnet	◒	Internet	network traffic	real-time, posteriori
Kuman et al. [37]	◒	✗	✗	✗	✗	Modbus, S7comm	◒	Internet	network traffic	–
Ferretti et al. [23]	◒	✗	✗	✗	◒	Modbus, S7comm, BACnet, EtherNet/IP	◒	VPN	network traffic	posteriori
HosTaGe [22]	◒	✗	◒	✗	✗	Modbus, S7comm	✗	Internet	network traffic	posteriori
Pliatsios et al. [20]	◒	✗	◒	✓	◒	Modbus	◒	VPN	network traffic	–
Honeyd+ [56]	◒	✗	✗	✗	✗	EtherNet/IP	◒	Internet	network traffic	–
THS [16]	◒	✗	◒	✗	✗	Modbus, S7comm, BACNet	◒	Internet	network traffic	–
CryPLH [21]	◒	✗	✗	◒	✗	S7comm	✗	Internet	network traffic	–
HoneyPhy [53]	◒	✓	✗	◒	◒	DNP3	✗	Internet, VPN	–	–
GasPot [35]	✗	✗	✗	✗	✗	–	✗	Internet	network traffic	–
Hilt et al. [29]	✓	✓	✓	✓	✓	–	✗	Internet	network traf., system, malware	posteriori
ICSpot [18]	◒	✓	✓	✓	✗	Modbus, S7comm	◒	Internet	network traffic	posteriori
Antonioli et al. [19]	✓	✓	✗	✓	✓	EtherNet/IP	✗	VPN	network traf., system, malware	–
Murillo et al. [13]	◒	✓	✗	✗	◒	EtherNet/IP	✗	VPN	–	–
MimePot [26]	◒	✓	✓	✗	◒	Modbus	✗	VPN	–	–
iHoney [48]	◒	✓	✓	✓	◒	Modbus, S7comm	◒	VPN	network traffic	real-time
HoneyPLC [41]	◒	✗	✗	✗	✗	S7comm	✓	Internet	network traf., ladder logic capture	–
HoneyICS	✓	✓	✓	✓	✓	Modbus, DNP3	✓	Internet, VPN	network traf., system, malware	real-time, posteriori

supervisory control network linking PLCs and HMI, and the field communication network to connect PLCs with the physical process.

Configurability. All reviewed honeypots, but [16, 38, 41], support only limited extensibility because they can impersonate only one or two PLC models. Similarly, most works, except for [22, 32, 38, 56], only support a limited number of ICS network protocols. This may significantly limit the ICS network they can emulate. HoneyICS currently can answer fingerprint requests from different PLC models e.g ABB, MicroLogix, and Siemens and supports the implementation of two of the most widely used protocols in ICS systems, namely Modbus and DNP3 (see paragraph "Extensions" in Section 7).

Scalability. All reviewed ICS honeypots, except for [18, 29, 53], have scalable designs as they adopt virtual resources and/or lightweight virtualization techniques. HoneyICS supports a high degree of application scalability because its implementation is based on container technology that greatly simplifies application deployment.

Honeypot Entry Point. The reviewed honeypots have been either exposed on the Internet [16, 21, 22, 32, 38, 55, 56] or protected via a VPN [13, 19, 20, 26]. Only [53] has been designed to support both kinds of entry points. Although exposing the honeypot on the Internet can provide an attacker with easy access, this limits the type of interactions (cf. Section 3.1). On the other hand, while VPN access allows the attacker to compromise the ICS network, the employment of a VPN might discourage the attacker as she has to go through an additional line of defense. HoneyICS supports both entry points to capture a larger spectrum of adversarial interactions.

Attack Monitoring. The reviewed honeypots widely differ in terms of supported data collection and analysis capabilities. Most of the existing honeypots either do not record any data about attackers' behavior [13, 26, 53, 55] or only maintain logs of network interactions [16, 18, 20, 22, 23, 32, 35, 37, 38, 48, 53, 56]. The only exceptions are [19, 29, 41]. Besides recording network interactions, [29] and [19] maintain logs of system events such as files created, processes launched, shell commands executed and malware dropped on the infected devices, while [41] captures the ladder logic code injected into the PLC. Among the reviewed works, only [18, 22, 23, 29, 32, 48] support some form of data analysis. The works in [32, 48] provide real-time analysis of the interactions with the honeypot and a visualization component that displays the data produced by the analysis. In contrast, [18, 22, 23] support various types of a posteriori analysis of the recorded network interactions [18, 23, 29] and attack pattern identification [22, 23, 29]. HoneyICS is able to capture detailed information about how an attacker interacts with the honeynet. Indeed, our monitoring system is able to capture not only the network traffic but also to record process activities and file system interactions such as malware injections. Moreover, HoneyICS provides a dashboard to visualize real-time statistics on network interactions and threat profiling based on the MITRE ATT&CK techniques.

stopped here.

7 DISCUSSION AND FUTURE WORK

We proposed and deployed HoneyICS: a high-interaction, physics-aware, scalable, and extensible honeynet for ICSs, equipped with an advanced monitoring system. We then conducted a number of experiments to evaluate its effectiveness. Below, we discuss security issues, (already) implemented extensions, and future work.

Security issues. While the high interaction requirement potentially exposes the honeypot to security risks, the containerization of its components allows for namespace isolation to limit interactions between containers and the OS kernel of the dashboard server. At the same time, the OS images of the containers are as simple as possible and free from unnecessary libraries. Thus, an attacker cannot manage to evade the reached container. To protect the management dashboard, access is regulated using a firewall, with a rule limiting the IP addresses allowed to interact with it. Moreover, access to the dashboard is also protected using strong authentication. To address ethical concerns, the egress traffic was blocked on all ports except for the ports used by the honeypots. Moreover, the scripts on the honeypots do not allow reaching cross-domain destinations.

Extensions. As OpenPLC can be configured to act as a DNP3 outstation, our framework can also support the industrial protocol DNP3 (vers. 3.0) [27]. This version of DNP3 does not include built-in security mechanisms, making it vulnerable to potential attackers who can forge, drop, or modify messages. A key difference between DNP3 and Modbus is that DNP3 has a more robust and sophisticated mechanism for handling data, including the ability to handle multiple data types and a more extensive range of data object types. Adopting DNP3 rather than Modbus requires only two changes in our honeynet: (i) in the brokers dealing with PLC communications; (ii) using Scada-LTS [4] instead of ScadaBR for the HMI.

We recall that OpenPLC [54] is a software PLC which executes programs that comply with the IEC 61131-3 standard [31]. Thus, we could leave the profile of OpenPLC "in clear" (i.e., without masking it with a different profile). This would allow an attacker to achieve *code injection*, i.e., changes on the PLC user program, via a brute force attack on the webserver of OpenPLC on port 8080.

Finally, although the implementation of our use case relies on software PLCs, both our architecture and our implementation support real *PLCs* (hardware) with almost no changes by simply connecting the PLC to: (i) a dedicated NGINX proxy to be part of the supervisory network, and (ii) a Raspberry Pi via its I/O ports; the Raspberry Pi will handle the signal exchange between the PLC and the real-time simulation of the plant.

Future Work. We plan to support other industrial network protocols, such as OPC-UA [52] and Ethernet/IP [49]. As suggested by Russo et al. [51], we intend to adopt lightweight simulation methods for physical processes such as interpolation, ODE simulation and scripting. Furthermore, following the example of HoneyPLC [41], we intend to support both the snap7 library [47], to implement an S7comm server listening on port 102 TCP, and the SNMP protocol, on port 161 UDP, to monitor devices connected to the network. Finally, we plan to perform an in-depth investigation of the network traffic collected using HoneyICS.

ACKNOWLEDGMENTS

Angelo Furfaro and Francesco Lupia have been partially supported by the "EMPHAsis Evolution" project (CUP J89J22003110005), funded by "Azione 1.1.5 POR Calabria FESR-FSE 2014-2020".

REFERENCES

[1] 2009. The ScadaBR project. https://www.scadabr.com.br/
[2] 2013. Kibana. https://www.elastic.co/kibana/
[3] 2016. S7comm - The Wireshark Wiki. https://wiki.wireshark.org/S7comm/
[4] 2021. Scada-LTS. https://scada-lts.org/
[5] 2022. GreyNoise, Inc. https://www.greynoise.io/
[6] 2023. AlienVault OTX. https://otx.alienvault.com/
[7] 2023. Aqua Tracee. https://www.aquasec.com/products/tracee/
[8] 2023. Elastic Search. https://www.elastic.co/
[9] 2023. Extended Berkeley Packet Filter. https://ebpf.io/
[10] 2023. Macvlan network. https://docs.docker.com/network/macvlan/
[11] 2023. MITRE ATT&CK Framework. https://attack.mitre.org/
[12] 2023. VirusTotal. https://www.virustotal.com/
[13] A. Murillo, L. Combita Alfonso, A. Gonzalez, S. Rueda, A. Cardenas, and N. Quijano. 2018. A Virtual Environment for Industrial Control Systems: A Nonlinear Use-Case in Attack Detection, Identification, and Response. In *ICSS*. 25–32.
[14] A. P. Mathur and N.O. Tippenhauer. 2016. SWaT: a water treatment testbed for research and training on ICS security. In *CySWater@CPSWeek*. IEEE, 31–36.
[15] A. Swales. 1999. Open Modbus/tcp specification. *Schneider Electric* 29 (1999).
[16] S. Abe, Y. Tanaka, Y. Uchida, and S. Horata. 2018. Developing Deception Network System with Traceback Honeypot in ICS Network. *JCMSI* 11 (2018), 372–379.
[17] C. Boettiger. 2015. An Introduction to Docker for Reproducible Research. *SIGOPS Oper. Syst. Rev. ACM* (2015), 71–79.
[18] M. Conti, F. Trolese, and F. Turrin. 2022. ICSpot: A High-Interaction Honeypot for Industrial Control Systems. In *ISNCC*. IEEE, 1–4.
[19] D. Antonioli, A. Agrawal, and N.O. Tippenhauer. 2016. Towards High-Interaction Virtual ICS Honeypots-in-a-Box. In *CPS-SPC*. ACM, 13–22.
[20] D. Pliatsios, P.G. Sarigiannidis, T. Liatifis, K. Rompolos, and I. Siniosoglou. 2019. A Novel and Interactive Industrial Control System Honeypot for Critical Smart Grid Infrastructure. In *IEEE CAMAD*. 1–6.
[21] D.I. Buza, F. Juhász, G. Miru, M. Félegyházi, and T. Holczer. 2014. CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot. In *Smart Grid Security*. Springer, 181–192.

[13, 16, 19, 21, 22, 26, 35, 38, 41, 53, 55, 56] h

[22] E. Vasilomanolakis, S. Srinivasa, C.G. Cordero, and M. Mühlhäuser. 2016. Multi-stage attack detection and signature generation with ICS honeypots. In *NOMS IEEE*. 1227–1232.
[23] P. Ferretti, M. Pogliani, and S. Zanero. 2019. Characterizing Background Noise in ICS Traffic Through a Set of Low Interaction Honeypots. In *CPS-SPC@CCS*. ACM, 51–61.
[24] A. Furfaro, L. Argento, A. Parise, and Piccolo. A. 2017. Using virtual environments for the assessment of cybersecurity issues in IoT scenarios. *Simul. Model. Pract. Theory* 73 (2017), 43–54.
[25] F. Furtado, S. Shrivastava, A. Mathur, and N. Goh. 2022. The Design of Cyber-Physical Exercises (CPXs). In *CyCon*. IEEE.
[26] G. Bernieri, M. Conti, and F. Pascucci. 2019. MimePot: a Model-based Honeypot for Industrial Control Networks. In *IEEE SMC*. 433–438.
[27] G. Clarke, D. Reynders, and E. Wright. 2004. *Practical Modern SCADA Protocols: DNP3, 60870.5 and Related Systems*. Newnes, Elsevier.
[28] G. Lyon. 1997. Nmap. https://nmap.org/
[29] S. Hilt, F. Maggi, C. Perine, L. Remorin, M. Rösler, and R. Vosseler. 2020. Caught in the Act: Running a Realistic Factory Honeypot to Capture Real Threats.
[30] Y. Huang, A. A. Cárdenas, S. Amin, Z. Lin, H. Tsai, and S. Sastry. 2009. Understanding the physical and economic consequences of attacks on control systems. *Int. J. Crit. Infrastructure Prot.* 2, 3 (2009), 73–83.
[31] International Electrotechnical Commission. 1993. Programmable controllers-Part 3 : Programming languages. *IEC 61131-3* (1993).
[32] J. Cao, W. Li, J. Li, and B. Li. 2018. *DiPot: A Distributed Industrial Honeypot System*. Springer, 300–309.
[33] J. Franco, A. Aris, B. Canberk, and A. Selcuk Uluagac. 2021. A Survey of Honeypots and Honeynets for Internet of Things, Industrial Internet of Things, and Cyber-Physical Systems. *IEEE Commun. Surv. Tutorials* (2021), 2351–2383.
[34] J. Searle. 2015. PLCscan. https://github.com/meeas/plcscan/
[35] K. Wilhoit and S. Hilt. 2015. The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems. In *Trend Micro*, Vol. 6. 3–13.
[36] M. Krotofil, K. Kursawe, and D. Gollmann. 2019. Securing Industrial Control Systems. In *Security and Privacy Trends in the Industrial Internet of Things*. Springer, 3–27.
[37] S. Kuman, S. Gros, and M. Mikuc. 2017. An experiment in using IMUNES and Conpot to emulate honeypot control networks. In *MIPRO*. IEEE, 1262–1268.
[38] L. Rist, J. Vestergaard, D. Haslinger, A. De Pasquale, and J. Smith. 2013. Conpot ICS/SCADA Honeypot. http://conpot.org/
[39] R. Lanotte, M. Merro, and A. Munteanu. 2023. Industrial Control Systems Security via Runtime nforcement. *ACM Trans. Priv. Secur.* 26, 1 (2023), 4:1–4:41.
[40] R. Lanotte, M. Merro, A. Munteanu, and L. Viganò. 2020. A Formal Approach to Physics-based Attacks in Cyber-physical Systems. *ACM Trans. Priv. Secur.* 23, 1 (2020), 3:1–3:41.
[41] E. López-Morales, C. Rubio, A. Doupé, Y. Shoshitaishvili, R. Wang, T. Bao, and G-H Ahn. 2020. *HoneyPLC: A Next-Generation Honeypot for Industrial Control Systems*. ACM, 279–291.
[42] M. Lucchese, M. Merro, F. Paci, and N. Zannone. 2023. Towards A High-Interaction Physics-Aware Honeynet for Industrial Control Systems. In *38th ACM/SIGAPP SAC (SAC '23)*. ACM, 76–79.
[43] M. Dodson, A.R. Beresford, and M. Vingaard. 2020. Using Global Honeypot Networks to Detect Targeted ICS Attacks. In *CyCon*. 275–291.
[44] M. Kerrisk. 27-08-2021. ld.so(8) - Linux manual page. https://man7.org/linux/man-pages/man8/ld.so.8.html
[45] MATLAB. 2021. version R2021a.
[46] N. Provos. 2003. Honeyd: A Virtual Honeypot Daemon (Extended Abstract). *DFN-CERT* 2 (2003).
[47] D. Nardella. 2013. Snap7 project. http://snap7.sourceforge.net/
[48] O. Navarro, S.A.J. Balbastre, and S. Beyer. 2019. Gathering Intelligence Through Realistic Industrial Control System Honeypots - A Real-World Industrial Experience Report. In *CRITIS*, Vol. 11260. Springer, 143–153.
[49] P. Brooks. 2001. Ethernet/IP-industrial protocol. In *ETFA*, Vol. 2. 505–514.
[50] R. Rajkumar, L. Lee, I. Sha, and J. A. Stankovic. 2010. Cyber-physical systems: the next computing revolution. In *DAC*. ACM, 731–736.
[51] E. Russo, G. Costa, G. Longo, A. Armando, and A. Merlo. 2023. LiDiTE: a Full-Fledged and Featherweight Digital Twin Framework. *IEEE Transactions on Dependable and Secure Computing* (2023), 1–14.
[52] S-H. Leitner and W. Mahnke. 2006. OPC UA–service-oriented architecture for industrial applications. *ABB Corporate Research Center* 48, 61-66 (2006), 22.
[53] S. Litchfield, D. Formby, J. D. Rogers, A. P. S. Meliopoulos, and R. A. Beyah. 2016. Rethinking the Honeypot for Cyber-Physical Systems. *IEEE Internet Comput.* 20 (2016), 9–17.
[54] T.R. Alves, M. Buratto, F.M. Souza, and T.V. Rodrigues. 2014. OpenPLC: An open source alternative to automation. In *IEEE GHTC*. 585–589.
[55] V. Pothamsetty and M. Franz. 2004. SCADA HoneyNet Project: Building Honeypots for Industrial Networks. (CIAG) Cisco Systems.
[56] M. Winn, M. Rice, S. Dunlap, J. Lopez Jr., and B. E. Mullins. 2015. Constructing cost-effective and targetable industrial control system honeypots for production networks. *Int. J. Crit. Infrastructure Prot.* 10 (2015), 47–58.